Лабораторная работа №4 по С#

Отношения классов.

Задание 1

Вариант 1

namespace вариант\_1\_задание\_1

{

internal class Facultet

{

public enum Allocation {Yes=1, No=0, Repeat=-1 }

public string Name { get; set; }

Kafedra[] Kafedras;

public Facultet()

{

Name = "Noname";

Kafedras = new Kafedra[1];

Kafedras[0] = new Kafedra("Noname");

}

public Facultet(string name):this()

{

Name = name;

}

public Facultet(string name, params string[] kafedrasNames) : this(name)

{

Kafedras = new Kafedra[kafedrasNames.Length];

for(int i = 0; i < kafedrasNames.Length; i++)

{

Kafedras[i]= new Kafedra(kafedrasNames[i]);

}

}

internal Allocation PrivKafedra(Auditoria auditoria)

{

foreach(Kafedra kafedra in Kafedras)

{

if(kafedra?.auditoria?.Name == auditoria?.Name)

{

return Allocation.Repeat;

}

}

for(int i = 0;i < Kafedras.Length; i++)

{

if (Kafedras[i].auditoria == null)

{

Kafedras[i].auditoria = auditoria;

auditoria.kafedra = Kafedras[i];

return Allocation.Yes;

}

}

return Allocation.No;

}

public override string? ToString()

{

string str = "Факультет ";

if (Name != null)

{

str += $"{Name}. Имеет кафедр - {Kafedras.Length}\n";

foreach (Kafedra kaf in Kafedras)

{

str += $"\t{kaf}\n";

}

}

return str;

}

}

}

namespace вариант\_1\_задание\_1

{

internal class Kafedra

{

public Auditoria auditoria=null;

public Kafedra(string name)

{

Name = name;

}

public string Name { get; set; }

public override string? ToString()

{

string str = "Кафедра ";

if (Name != null)

{

str += $"{Name}" +

(auditoria == null ? " не имеет аудитории" : $" находится в аудитории {auditoria.Name}");

}

return str;

}

}

}

namespace вариант\_1\_задание\_1

{

internal class Auditoria

{

public string Name { get; set; }

public int KolMest { get; set; }

public Kafedra kafedra;

public Auditoria():this("123", 20)

{

}

public Auditoria(string name, int kolMest)

{

Name = name;

KolMest = kolMest;

}

public Facultet.Allocation PrivRequest(Facultet facultet)

{

return facultet.PrivKafedra(this);

}

public override string? ToString()

{

string str = "Аудитория ";

if (Name != null)

{

str += $"{Name} с количеством мест {KolMest}"+

(kafedra==null? ", которая не привязана к кафедре":$",которая привязана к кафедре {kafedra.Name}");

}

return str;

}

}

}

using вариант\_1\_задание\_1;

Facultet facultet1 = new Facultet();

Console.WriteLine(facultet1);

Facultet facultet2 = new Facultet("УЭФ", "БУП", "БУС");

Console.WriteLine(facultet2);

Auditoria auditoria1 = new Auditoria("407", 68);

Console.WriteLine(auditoria1);

Auditoria auditoria2 = new Auditoria("510", 85);

Console.WriteLine(auditoria2);

Auditoria auditoria3 = new Auditoria("633", 74);

Console.WriteLine(auditoria3);

Console.WriteLine();

Console.WriteLine($"требуется привязать аудиторию {auditoria1} к кафедре на факультете {facultet2}");

Console.WriteLine(auditoria1.PrivRequest(facultet2));

Console.WriteLine(facultet2);

Console.WriteLine($"требуется привязать аудиторию {auditoria1} к кафедре на факультете {facultet2}");

Console.WriteLine(auditoria1.PrivRequest(facultet2));

Console.WriteLine(facultet2);

Console.WriteLine();

Console.WriteLine($"требуется привязать аудиторию {auditoria2} к кафедре на факультете {facultet2}");

Console.WriteLine(auditoria2.PrivRequest(facultet2));

Console.WriteLine(facultet2);

Console.WriteLine();

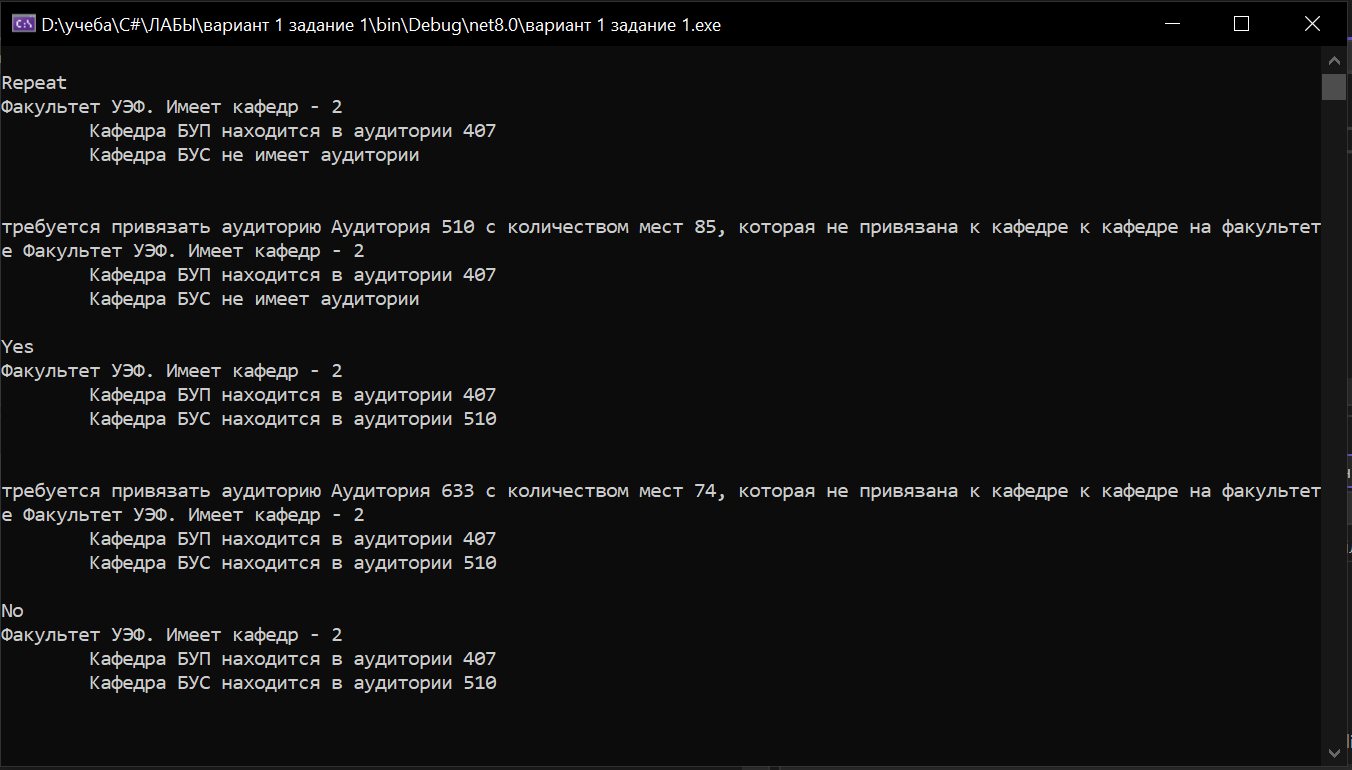
Console.WriteLine($"требуется привязать аудиторию {auditoria3} к кафедре на факультете {facultet2}");

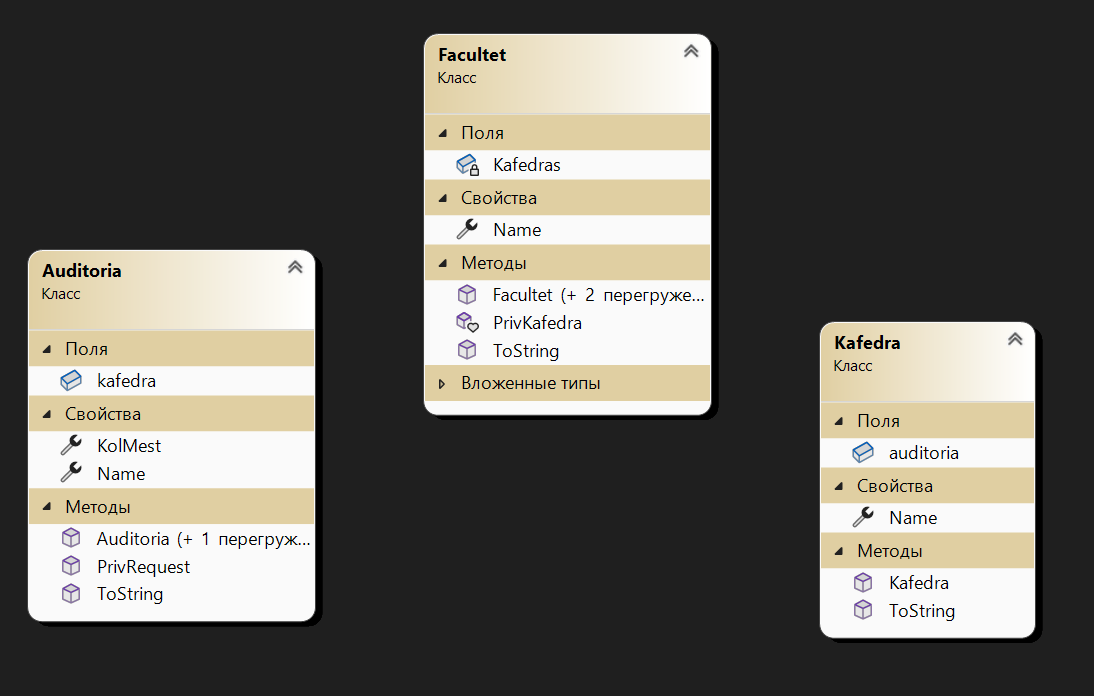
Console.WriteLine(auditoria3.PrivRequest(facultet2));

Console.WriteLine(facultet2);

Console.WriteLine();

Console.ReadKey();





1

1

1

M

Вариант 2

namespace задание\_1\_вариант\_2

{

internal class Garag

{

public enum Allocation {Yes=1,No=0,Repeat=-1}

public string Address { get; set; }

public Auto[] autos;

public Garag()

{

Address = "Noadress";

autos = new Auto[1];

autos[0]= new Auto("Noname");

}

public Garag(string address):this()

{

Address = address;

}

public Garag(string address, params string[] autosNames) : this(address)

{

autos=new Auto[autosNames.Length];

for(int i = 0; i < autosNames.Length; i++)

{

autos[i]=new Auto(autosNames[i]);

}

}

public Allocation AutoRequest(Driver driver)

{

foreach(Auto auto in autos)

{

if(auto?.driver?.Name == driver?.Name)

{

return Allocation.Repeat;

}

}

for(int i = 0;i < autos.Length;i++)

{

if (autos[i].driver == null)

{autos[i].driver = driver;

driver.auto = autos[i];

return Allocation.Yes;

}

}

return Allocation.No;

}

public override string? ToString()

{

string str = $"Гараж по адрессу {Address}. Машин в гараже - {autos.Length}.\n";

foreach(Auto aut in autos)

{

str += $"\t{aut}\n";

}

return str;

}

}

}

namespace задание\_1\_вариант\_2

{

internal class Auto

{

public string Name { get; set; }

public Driver driver;

public Auto():this("Noname")

{

}

public Auto(string name)

{

Name = name;

}

public override string? ToString()

{

string str = "Автомобиль ";

if (Name != null)

{

str += $"{Name}" +

(driver == null ? " без водителя" : $" с водителем {driver.Name}");

}

return str;

}

}

}

namespace задание\_1\_вариант\_2

{

internal class Driver

{

public string Name { get; set; }

public Auto auto;

public Driver():this("Noname")

{

}

public Driver(string name)

{

Name = name;

}

public Garag.Allocation DriveRequest(Garag garag)

{

return garag.AutoRequest(this);

}

public override string? ToString()

{

string str = "Водитель ";

if(Name != null)

{

str += $"{Name}" +

(auto == null ? " свободен" : $"сядет за руль {auto.Name}");

}

return str;

}

}

}

using задание\_1\_вариант\_2;

Garag garag1 = new Garag();

Console.WriteLine(garag1);

Garag garag2 = new Garag("Мстиславца 4", "Ауди", "БМВ", "Мерседес");

Console.WriteLine(garag2);

Driver driver1 = new Driver("John");

Driver driver2 = new Driver("Tim");

Driver driver3 = new Driver("Tony");

Driver driver4 = new Driver("Jackson");

Console.WriteLine($"Водитель по имени {driver1.Name} сядет за руль одной из машин в гараже по адресу {garag2.Address}");

Console.WriteLine(driver1.DriveRequest(garag2));

Console.WriteLine(garag2);

Console.WriteLine($"Водитель по имени {driver1.Name} сядет за руль одной из машин в гараже по адресу {garag2.Address}");

Console.WriteLine(driver1.DriveRequest(garag2));

Console.WriteLine(garag2);

Console.WriteLine($"Водитель по имени {driver2.Name} сядет за руль одной из машин в гараже по адресу {garag2.Address}");

Console.WriteLine(driver2.DriveRequest(garag2));

Console.WriteLine(garag2);

Console.WriteLine($"Водитель по имени {driver3.Name} сядет за руль одной из машин в гараже по адресу {garag2.Address}");

Console.WriteLine(driver3.DriveRequest(garag2));

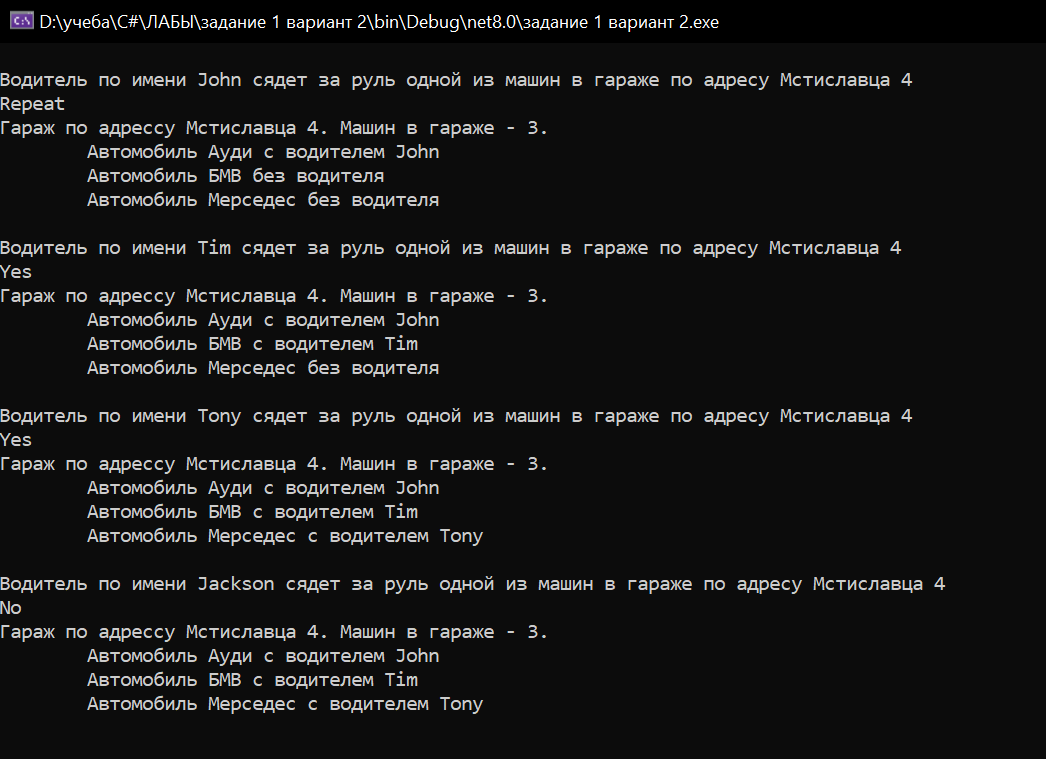
Console.WriteLine(garag2);

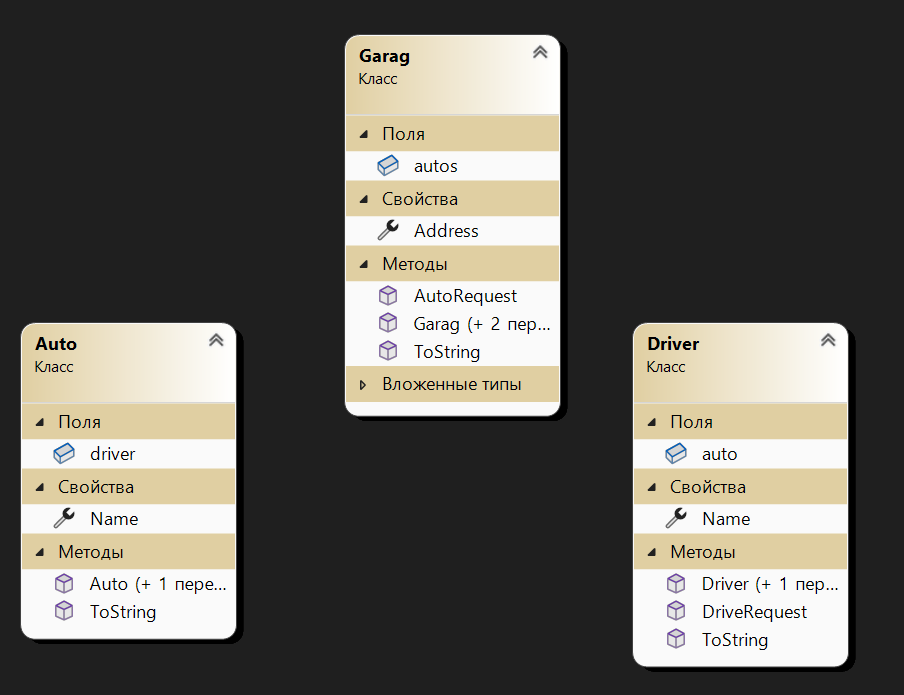
Console.WriteLine($"Водитель по имени {driver4.Name} сядет за руль одной из машин в гараже по адресу {garag2.Address}");

Console.WriteLine(driver4.DriveRequest(garag2));

Console.WriteLine(garag2);

Console.ReadKey();





1

М

1

1

Вариант 4

namespace вариант\_4\_задание\_1

{

internal class Port

{

public enum Allocation {Yes=1,No=0,Repeat=-1 }

public string Name { get; set; }

Tanker[] tankers;

public Port()

{

Name = "Noname";

tankers = new Tanker[1];

tankers[0] = new Tanker("Noname");

}

public Port(string name)

{

Name = name;

}

public Port(string name, params string[] tankersNames) : this(name)

{

tankers=new Tanker[tankersNames.Length];

for(int i = 0; i < tankersNames.Length; i++)

{

tankers[i] = new Tanker(tankersNames[i]);

}

}

public Allocation TankerRequest(Gruz gruz)

{

foreach(var tanker in tankers)

{

if (tanker?.gruz?.Name == gruz?.Name)

{

return Allocation.Repeat;

}

}

for(int i = 0;i < tankers.Length; i++)

{

if (tankers[i].gruz == null)

{

tankers[i].gruz = gruz;

gruz.tanker=tankers[i];

return Allocation.Yes;

}

}

return Allocation.No;

}

public override string? ToString()

{

string str = $"Порт {Name}. Стоит танкеров на загрузку - {tankers.Length}\n";

foreach(var tanker in tankers)

{

str += $"\t{tanker}\n";

}

return str;

}

}

}

namespace вариант\_4\_задание\_1

{

internal class Tanker

{

public string Name { get; set; }

public Gruz gruz;

public Tanker():this("Noname")

{

}

public Tanker(string name)

{

Name = name;

}

public override string? ToString()

{

string str = "Танкер ";

if(Name != null)

{

str += $"{Name} " +

(gruz == null ? "без груза" : $"с грузом {gruz.Name}");

}

return str;

}

}

}

namespace вариант\_4\_задание\_1

{

internal class Gruz

{

public Tanker tanker = null;

public string Name { get; set; }

public Gruz():this("Noname")

{

}

public Gruz(string name)

{

Name = name;

}

public Port.Allocation ZagruzRequest(Port port)

{

return port.TankerRequest(this);

}

public override string? ToString()

{

string str = "Груз ";

if (Name != null)

{

str += $"{Name} " +

(tanker == null ? "не погружен на танкер" : $"погружен на танкер {tanker.Name}");

}

return str;

}

}

}

using вариант\_4\_задание\_1;

Port port1 =new Port();

Console.WriteLine(port1);

Port port2 =new Port("Hell", "JustDoIt", "Rise", "TheDarkKnight");

Gruz gruz1 = new Gruz("Игрушки");

Gruz gruz2 = new Gruz("Одежда");

Gruz gruz3 = new Gruz("Телефоны");

Gruz gruz4 = new Gruz("Компьютеры");

Console.WriteLine($"требуется танкер в порту {port2.Name} под загрузку груза {gruz1.Name}");

Console.WriteLine(gruz1.ZagruzRequest(port2));

Console.WriteLine(port2);

Console.WriteLine($"требуется танкер в порту {port2.Name} под загрузку груза {gruz1.Name}");

Console.WriteLine(gruz1.ZagruzRequest(port2));

Console.WriteLine(port2);

Console.WriteLine($"требуется танкер в порту {port2.Name} под загрузку груза {gruz2.Name}");

Console.WriteLine(gruz2.ZagruzRequest(port2));

Console.WriteLine(port2);

Console.WriteLine($"требуется танкер в порту {port2.Name} под загрузку груза {gruz3.Name}");

Console.WriteLine(gruz3.ZagruzRequest(port2));

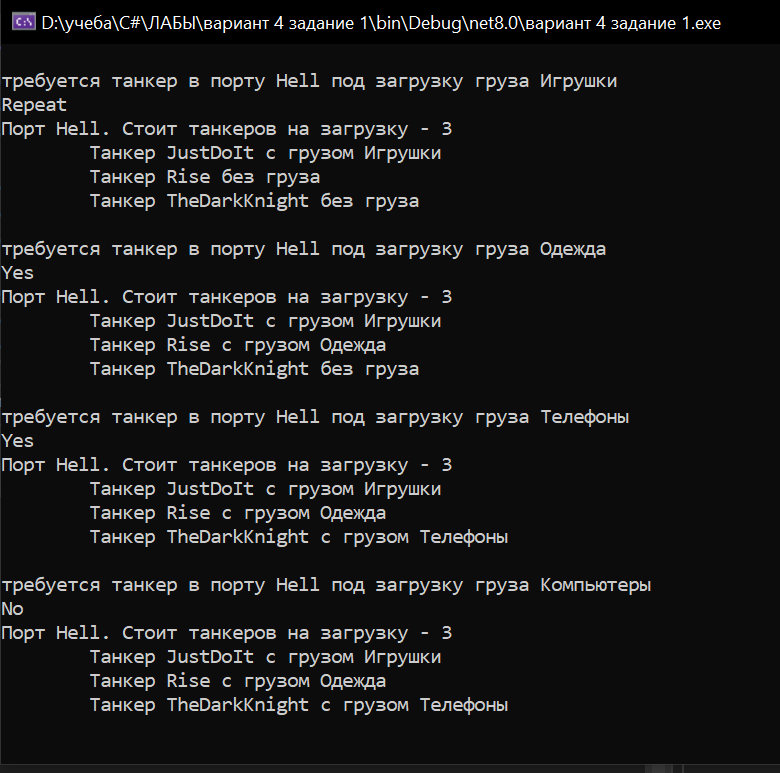
Console.WriteLine(port2);

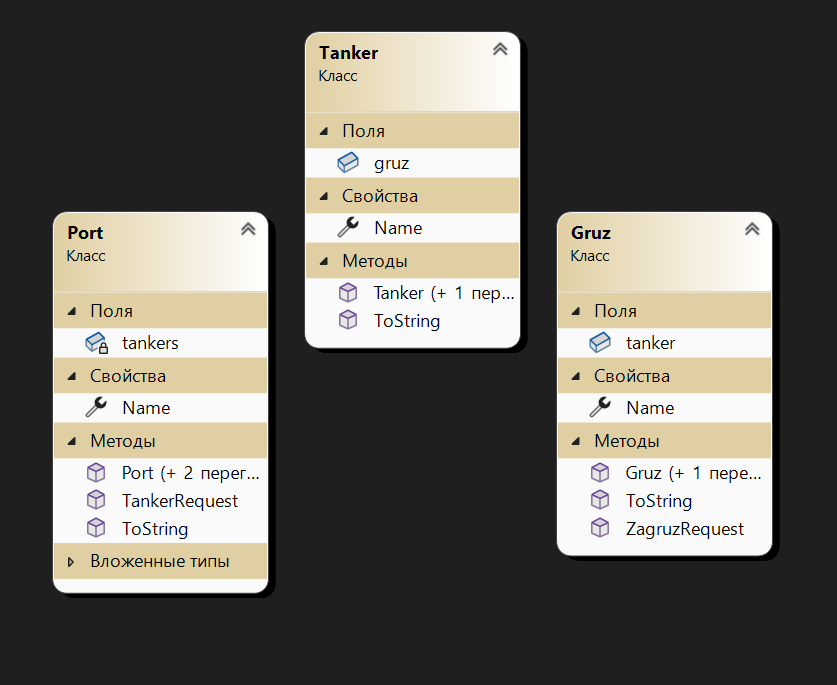
Console.WriteLine($"требуется танкер в порту {port2.Name} под загрузку груза {gruz4.Name}");

Console.WriteLine(gruz4.ZagruzRequest(port2));

Console.WriteLine(port2);

Console.ReadKey();





1

1

M

1

Вариант 5

namespace вариант\_5\_задание\_1

{

internal class Hotel

{

public enum Allocation { Yes=1, No=0, Repeat=-1}

public string Name { get; set; }

Room[] rooms;

public Hotel()

{

Name = "Noname";

rooms=new Room[1];

rooms[0] = new Room(000);

}

public Hotel(string name):this()

{

Name = name;

}

public Hotel(string name, params int[] roomsNumbers) : this(name)

{

rooms=new Room[roomsNumbers.Length];

for(int i=0; i<roomsNumbers.Length; i++)

{

rooms[i] = new Room(roomsNumbers[i]);

}

}

public Allocation ClientRequest(Client client)

{

foreach(Room room in rooms)

{

if (room?.client?.Name == client.Name)

{

return Allocation.Repeat;

}

}

for(int i=0; i < rooms.Length; i++)

{

if (rooms[i].client == null)

{

rooms[i].client = client;

client.room = rooms[i];

return Allocation.Yes;

}

}

return Allocation.No;

}

public override string? ToString()

{

string str = $"Гостиница {Name}. C количеством номеров для жильцов - {rooms.Length}\n";

foreach (Room room in rooms)

{

str += $"\t{room}\n";

}

return str;

}

}

}

namespace вариант\_5\_задание\_1

{

internal class Room

{

public Client client;

public Room():this(000)

{

}

public Room(int number)

{

Number = number;

}

public int Number { get; set; }

public override string? ToString()

{

string str = $"Номер {Number}" +

(client == null ? ", в который никто не заселен" : $", в который заселен {client.Name}");

return str;

}

}

}

namespace вариант\_5\_задание\_1

{

internal class Client

{

public Room room=null;

public Client():this("Noname")

{

}

public Client(string name)

{

Name = name;

}

public string Name { get; set; }

public Hotel.Allocation RoomRequest(Hotel hotel)

{

return hotel.ClientRequest(this);

}

public override string? ToString()

{

string str = "Клиент ";

if (Name != null)

{

str += $"{Name} " +

(room == null ? "не заселен" : $"зеселен в номер {room.Number}");

}

return str;

}

}

}

using вариант\_5\_задание\_1;

Hotel hotel1 = new Hotel();

Console.WriteLine(hotel1);

Hotel hotel2 = new Hotel("Mercy", 123, 345, 567);

Client client1 = new Client("Jonh");

Client client2 = new Client("Don");

Client client3 = new Client("Ron");

Client client4 = new Client("Ken");

Console.WriteLine($"Клиент {client1.Name} заселится в гостиницу {hotel2}");

Console.WriteLine(client1.RoomRequest(hotel2));

Console.WriteLine(hotel2);

Console.WriteLine($"Клиент {client1.Name} заселится в гостиницу {hotel2}");

Console.WriteLine(client1.RoomRequest(hotel2));

Console.WriteLine(hotel2);

Console.WriteLine($"Клиент {client2.Name} заселится в гостиницу {hotel2}");

Console.WriteLine(client2.RoomRequest(hotel2));

Console.WriteLine(hotel2);

Console.WriteLine($"Клиент {client3.Name} заселится в гостиницу {hotel2}");

Console.WriteLine(client3.RoomRequest(hotel2));

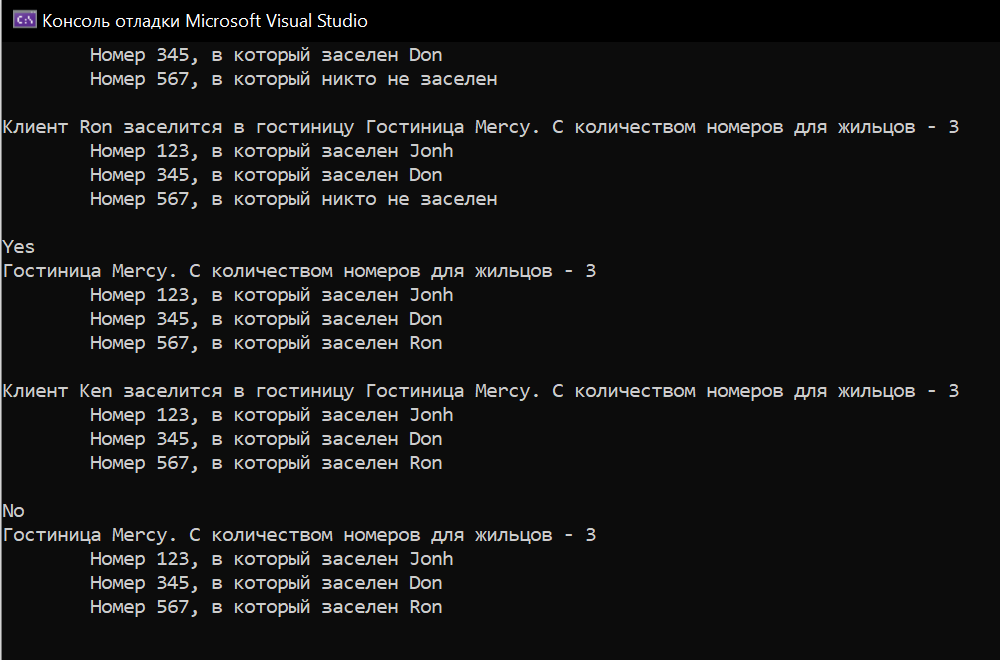
Console.WriteLine(hotel2);

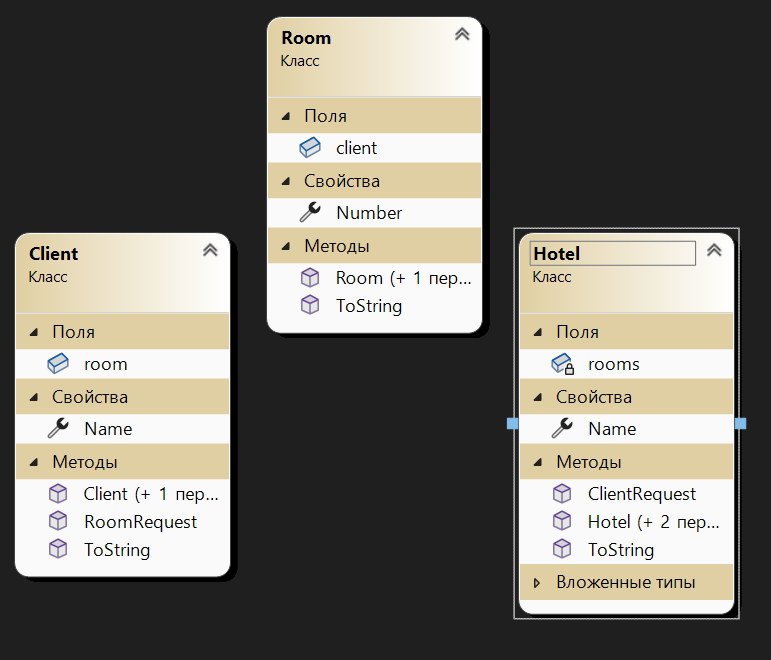
Console.WriteLine($"Клиент {client4.Name} заселится в гостиницу {hotel2}");

Console.WriteLine(client4.RoomRequest(hotel2));

Console.WriteLine(hotel2);

Console.ReadKey();





1

1

1

M

Вариант 6

namespace Вариант\_6\_задание\_1

{

internal class Town

{

public enum Allocation {Yes=1,No=0,Repeat=-1 }

public string Name { get; set; }

Sight[] sights;

public Town()

{

Name = "Noname";

sights = new Sight[1];

sights[0]= new Sight("Noname");

}

public Town(string name, params Sight[] sights)

{

Name = name;

this.sights = sights;

}

public Allocation GidPriv(Gid gid)

{

foreach (var s in sights)

{

if (s?.gid?.Name == gid.Name)

{

return Allocation.Repeat;

}

}

for(int i=0; i<sights.Length; i++)

{

if (sights[i].gid == null)

{

sights[i].gid = gid;

gid.sight = sights[i];

return Allocation.Yes;

}

}

return Allocation.No;

}

public override string? ToString()

{

string str = $"Город {Name}. Количество достопр-стей - {sights.Length}\n";

for(int i = 0; i < sights.Length; i++)

{

str+= $"\t{sights[i].ToString()}\n";

}

return str ;

}

}

}

namespace Вариант\_6\_задание\_1

{

internal class Sight

{

public string Name { get; set; }

public Gid gid;

public Sight():this("Noname")

{

}

public Sight(string name)

{

Name = name;

}

public override string? ToString()

{

string str = "Достопри-сть ";

if(Name != null)

{

str += $"{Name} " +

(gid == null ? "не имеет гида" : $"имеет гида {gid.Name}");

}

return str;

}

}

}

namespace Вариант\_6\_задание\_1

{

internal class Gid

{

public string Name { get; set; }

public Sight sight;

public Gid():this("Noname")

{

}

public Gid(string name)

{

Name = name;

}

public Town.Allocation SightPriv(Town town)

{

return town.GidPriv(this);

}

public override string? ToString()

{

string str = "Гид ";

if (Name != null)

{

str += $"{Name} " +

(sight == null ? "не привязан к достопри-сти" : $"привязан к дост-сти {sight.Name}");

}

return str;

}

}

}

using Вариант\_6\_задание\_1;

Town town1 = new Town();

Console.WriteLine(town1);

Sight sight1 = new Sight("Национальная библиотека");

Sight sight2 = new Sight("Башни-близнецы");

Sight sight3 = new Sight("Мост");

Town town2 = new Town("Минск", sight1,sight2,sight3);

Console.WriteLine(town2);

Gid gid1 = new Gid("John");

Gid gid2 = new Gid("Tony");

Gid gid3 = new Gid("Edward");

Gid gid4 = new Gid("Steven");

Console.WriteLine($"Гиду по имени {gid1.Name} для пров-я экскурсии требуется достопримечательность в городе {town2.Name}");

Console.WriteLine(gid1.SightPriv(town2));

Console.WriteLine(town2);

Console.WriteLine($"Гиду по имени {gid1.Name} для пров-я экскурсии требуется достопримечательность в городе {town2.Name}");

Console.WriteLine(gid1.SightPriv(town2));

Console.WriteLine(town2);

Console.WriteLine($"Гиду по имени {gid2.Name} для пров-я экскурсии требуется достопримечательность в городе {town2.Name}");

Console.WriteLine(gid2.SightPriv(town2));

Console.WriteLine(town2);

Console.WriteLine($"Гиду по имени {gid3.Name} для пров-я экскурсии требуется достопримечательность в городе {town2.Name}");

Console.WriteLine(gid3.SightPriv(town2));

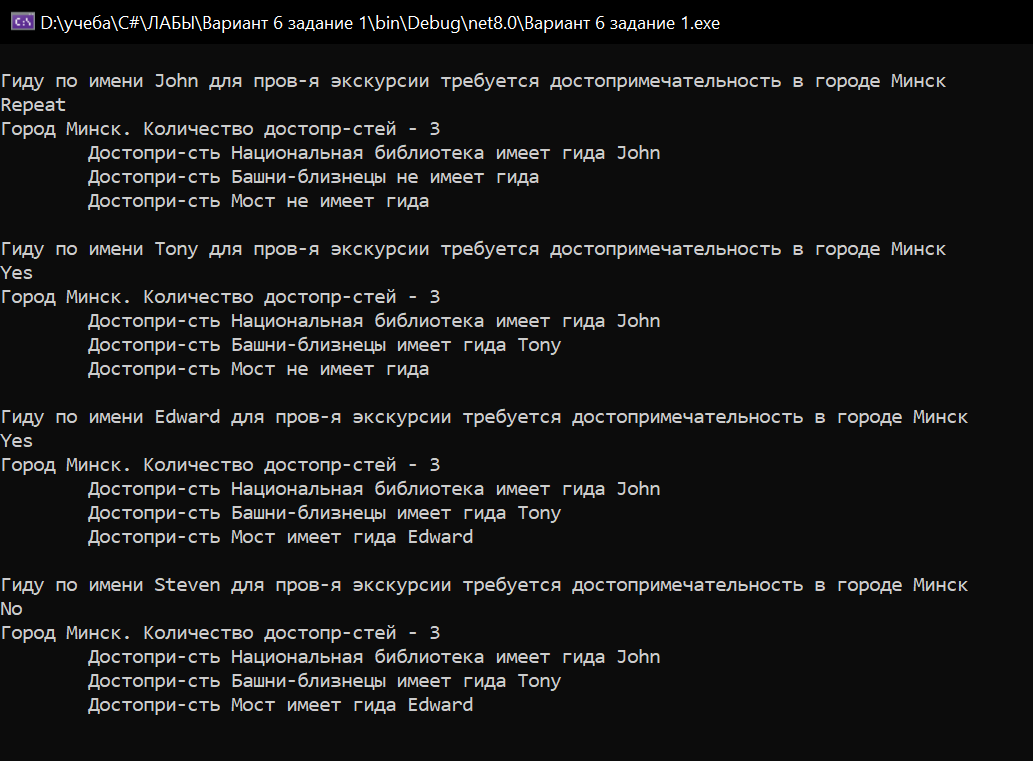
Console.WriteLine(town2);

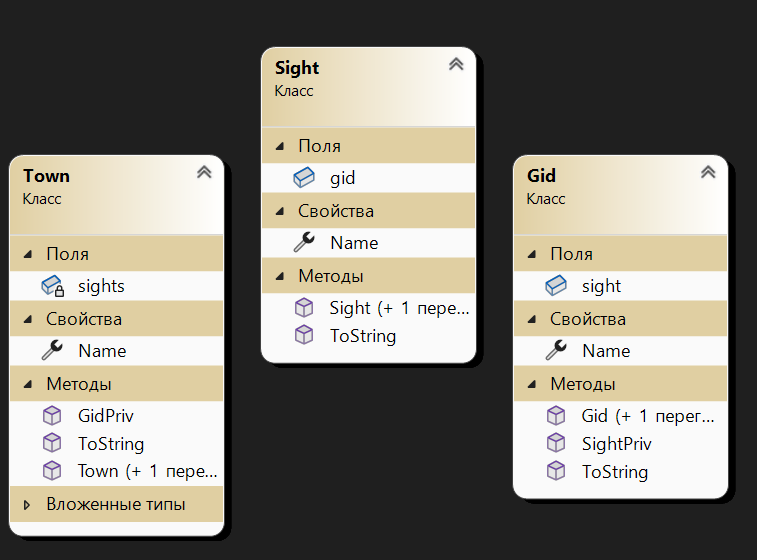
Console.WriteLine($"Гиду по имени {gid4.Name} для пров-я экскурсии требуется достопримечательность в городе {town2.Name}");

Console.WriteLine(gid4.SightPriv(town2));

Console.WriteLine(town2);

Console.ReadKey();





1

1

1

M

Вариант 7

namespace Вариант\_7\_задание\_1

{

internal class Shop

{

public enum Allocation {Yes=1,No=0,Repeat=-1 }

public string Name { get; set; }

Product[] products=new Product[0];

public Shop():this("Noname")

{

}

public Shop(string name)

{

Name = name;

}

public Shop(string name, params Product[] products) : this(name)

{

this.products = products;

}

public Allocation SalesRequest(Salesman salesman)

{

foreach (var item in products)

{

if (salesman?.product?.Name == item.Name)

{

return Allocation.Repeat;

}

}

for(int i=0; i<products.Length; i++)

{

if (products[i].salesman == null)

{

products[i].salesman = salesman;

salesman.product= products[i];

return Allocation.Yes;

}

}

return Allocation.No;

}

public override string? ToString()

{

if (products.Length == 0) return $"Магазин {Name} пустой";

else

{

string str = "Магазин ";

if(Name!=null)

{

str += $"{Name} имеет ассортимент. Видов продукции - {products.Length}\n";

foreach(Product p in products)

{

str += $"\t{p.ToString()}\n";

}

}

return str;

}

}

}

}

namespace Вариант\_7\_задание\_1

{

internal class Salesman

{

public string Name { get; set; }

public Product product;

public Salesman():this("Noname")

{

}

public Salesman(string name)

{

Name = name;

}

public Shop.Allocation ProductSales(Shop shop)

{

return shop.SalesRequest(this);

}

public override string? ToString()

{

string str = "Продавец ";

if(Name != null)

{

str += $"{Name} " +

(product == null ? "ничего не продает" : $"продает товар {product.Name}");

}

return str;

}

}

}

namespace Вариант\_7\_задание\_1

{

internal class Product

{

public string Name { get; set; }

public Salesman salesman;

public Product():this("Noname")

{

}

public Product(string name)

{

Name = name;

}

public override string? ToString()

{

string str = "Товар ";

if (Name != null)

{

str += $"{Name} " +

(salesman == null ? "никем не продается" : $"продается продавцом {salesman.Name}");

}

return str;

}

}

}

using Вариант\_7\_задание\_1;

Shop shop= new Shop();

Console.WriteLine(shop);

Product product1= new Product("Игрушки");

Product product2 = new Product("Техника");

Product product3 = new Product("Бытовые товары");

Shop shop2= new Shop("Марта",product1,product2,product3);

Console.WriteLine(shop2);

Salesman salesman1= new Salesman("Edward");

Salesman salesman2 = new Salesman("Tim");

Salesman salesman3 = new Salesman("Jack");

Salesman salesman4 = new Salesman("Tom");

Console.WriteLine($"Продавец {salesman1.Name} устраивается в магазин {shop2.Name}");

Console.WriteLine(salesman1.ProductSales(shop2));

Console.WriteLine(shop2);

Console.WriteLine($"Продавец {salesman1.Name} устраивается в магазин {shop2.Name}");

Console.WriteLine(salesman1.ProductSales(shop2));

Console.WriteLine(shop2);

Console.WriteLine($"Продавец {salesman2.Name} устраивается в магазин {shop2.Name}");

Console.WriteLine(salesman2.ProductSales(shop2));

Console.WriteLine(shop2);

Console.WriteLine($"Продавец {salesman3.Name} устраивается в магазин {shop2.Name}");

Console.WriteLine(salesman3.ProductSales(shop2));

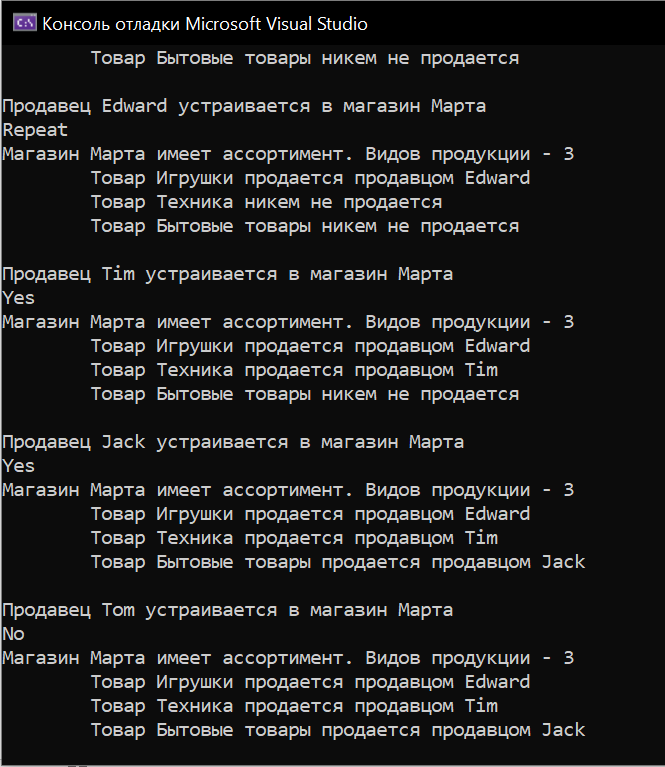
Console.WriteLine(shop2);

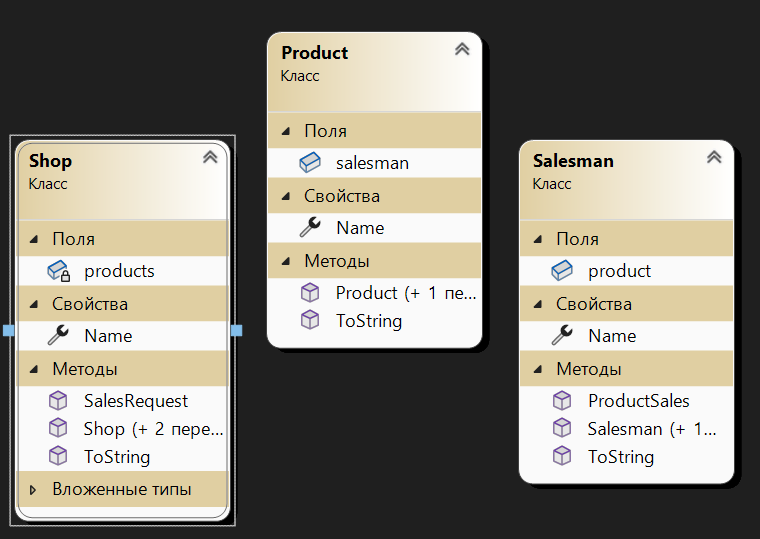
Console.WriteLine($"Продавец {salesman4.Name} устраивается в магазин {shop2.Name}");

Console.WriteLine(salesman4.ProductSales(shop2));

Console.WriteLine(shop2);

Console.ReadKey();





1

1

1

M

Вариант 8

namespace Вариант\_8\_задание\_1

{

internal class Railway

{

public enum Allocation {Yes=1, No=0,Repeat=-1 }

public string Name { get; set; }

Train[] trains=new Train[0];

public Railway():this("Noname")

{

}

public Railway(string name)

{

Name = name;

}

public Railway(string name, params Train[] trains) : this(name)

{

this.trains = trains;

}

public Allocation Request(Cargo cargo)

{

foreach (var tra in trains)

{

if (tra?. cargo?.Name == cargo.Name)

{

return Allocation.Repeat;

}

}

for(int i=0; i<trains.Length; i++)

{

if (trains[i].cargo == null)

{

trains[i].cargo = cargo;

cargo.train = trains[i];

return Allocation.Yes;

}

}

return Allocation.No;

}

public override string? ToString()

{

if (trains.Length == 0)

{

string str = "Вокзал ";

if(Name != null)

str +=$"{Name} пустой, т.е. без поездов на загрузку";

return str;

}

else

{

string str = "Вокзал ";

if (Name != null)

str += $"{Name}. Поездов на загрузку - {trains.Length}\n";

foreach(Train t in trains)

{

str += $"\t{t.ToString()}\n";

}

return str;

}

}

}

}

namespace Вариант\_8\_задание\_1

{

internal class Train

{

public string Name { get; set; }

public Cargo cargo;

public Train():this("Noname")

{

}

public Train(string name)

{

Name = name;

}

public override string? ToString()

{

string str = "Поезд ";

if (Name != null)

{

str += $"{Name} " +

(cargo == null ? "не загружен" : $"загружен грузом {cargo.Name}");

}

return str;

}

}

}

namespace Вариант\_8\_задание\_1

{

internal class Cargo

{

public string Name { get; set; }

public Train train;

public Cargo():this("Noname")

{

}

public Cargo(string name)

{

Name = name;

}

public Railway.Allocation TrainRequest(Railway railway)

{

return railway.Request(this);

}

public override string? ToString()

{

string str = "Груз ";

if (Name != null)

{

str += $"{Name} " +

(train == null ? "не загружен" : $"загружен в поезд {train.Name}");

}

return str;

}

}

}

using Вариант\_8\_задание\_1;

Railway railway = new Railway();

Console.WriteLine(railway);

Train train1 = new Train("Henry");

Train train2 = new Train("Lom");

Railway railway1 = new Railway("Garden", train1,train2);

Console.WriteLine(railway1);

Cargo cargo1 = new Cargo("Игрушки");

Cargo cargo2 = new Cargo("Варежки");

Cargo cargo3 = new Cargo("Машинки");

Console.WriteLine($"Груз {cargo1.Name} готов к загрузке в свободный поезд на вокзале {railway1.Name}");

Console.WriteLine(cargo1.TrainRequest(railway1));

Console.WriteLine(railway1);

Console.WriteLine($"Груз {cargo1.Name} готов к загрузке в свободный поезд на вокзале {railway1.Name}");

Console.WriteLine(cargo1.TrainRequest(railway1));

Console.WriteLine(railway1);

Console.WriteLine($"Груз {cargo2.Name} готов к загрузке в свободный поезд на вокзале {railway1.Name}");

Console.WriteLine(cargo2.TrainRequest(railway1));

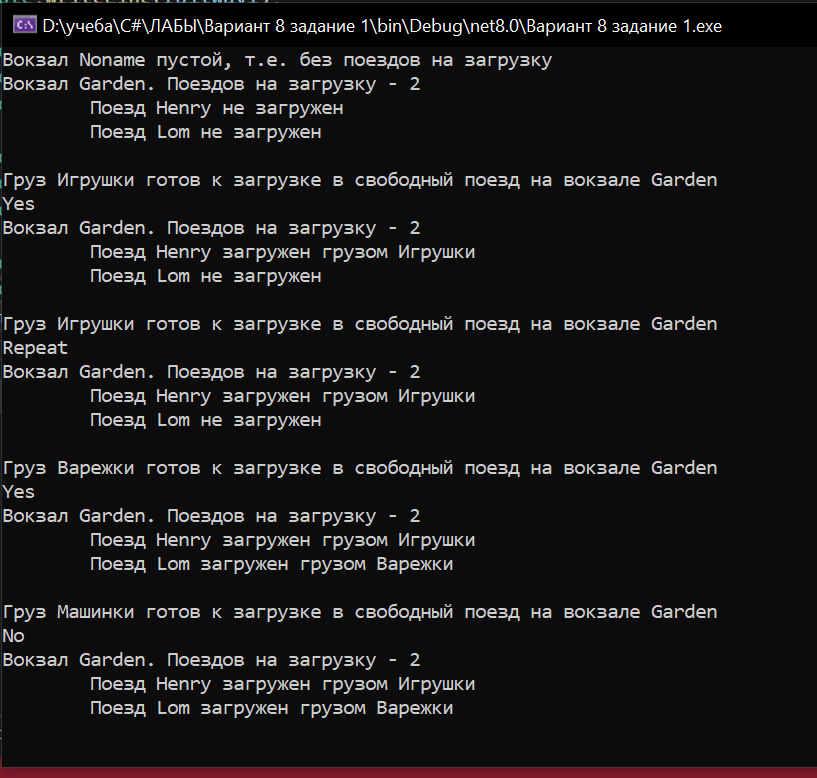
Console.WriteLine(railway1);

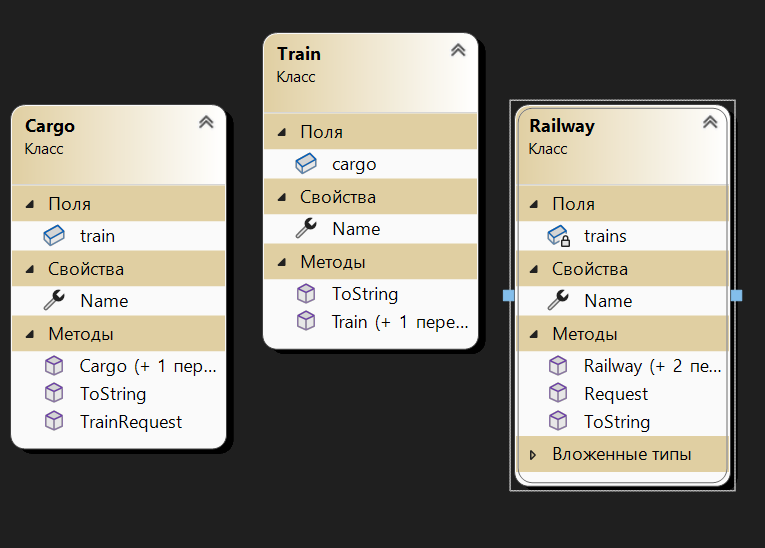
Console.WriteLine($"Груз {cargo3.Name} готов к загрузке в свободный поезд на вокзале {railway1.Name}");

Console.WriteLine(cargo3.TrainRequest(railway1));

Console.WriteLine(railway1);

Console.ReadKey();





1

1

1

M

Вариант 9

namespace Вариант\_9\_задание\_1

{

internal class Operator

{

public enum Allocation {Yes=1,No=0,Repeat=-1 }

public string Name { get; set; }

Plan[] plans;

public Operator()

{

Name = "Noname";

plans = new Plan[1];

plans[0]= new Plan("Noname");

}

public Operator(string name)

{

Name = name;

}

public Operator(string name, params string[] plansNames) : this(name)

{

plans=new Plan[plansNames.Length];

for(int i = 0; i < plansNames.Length; i++)

{

plans[i]= new Plan(plansNames[i]);

}

}

public Allocation Request(Abonent abonent)

{

foreach(Plan plan in plans)

{

if(plan?.abonent?.Name == abonent?.Name)

{

return Allocation.Repeat;

}

}

for(int i = 0;i < plans.Length;i++)

{

if (plans[i].abonent == null)

{

plans[i].abonent = abonent;

abonent.plan = plans[i];

return Allocation.Yes;

}

}

return Allocation.No;

}

public override string? ToString()

{

string str = "Интернет-оператор ";

if(Name!=null)

{

str += $"{Name}. Количество тарифных планов - {plans.Length}\n";

foreach(Plan plan in plans)

{

str += $"\t{plan.ToString()}\n";

}

}

return str;

}

}

}

namespace Вариант\_9\_задание\_1

{

internal class Abonent

{

public string Name { get; set; }

public Plan plan;

public Abonent():this("Noname")

{

}

public Abonent(string name)

{

Name = name;

}

public Operator.Allocation PlanRequest(Operator oper)

{

return oper.Request(this);

}

public override string? ToString()

{

string str = "Абонент ";

if (Name != null)

{

str += $"{Name} " +

(plan == null ? ", который не подписан на тарифный план" : $"подписан на тарифный план {plan.Name}");

}

return str;

}

}

}

namespace Вариант\_9\_задание\_1

{

internal class Plan

{

public string Name { get; set; }

public Abonent abonent;

public Plan():this("Noname")

{

}

public Plan(string name)

{

Name = name;

}

public override string? ToString()

{

string str = "Тарифный план ";

if (Name != null)

{

str += $"{Name} " +

(abonent == null ? ", на который никто не подписан" : $"подписан абонент {abonent.Name}");

}

return str;

}

}

}

using Вариант\_9\_задание\_1;

Operator oper = new Operator();

Console.WriteLine(oper);

Operator oper1 = new Operator("MTC","L","XXL");

Console.WriteLine(oper1);

Abonent abonent1 = new Abonent("Nick");

Abonent abonent2 = new Abonent("Jim");

Abonent abonent3 = new Abonent("Vlad");

Console.WriteLine($"Абонент {abonent1.Name} хочет получить услугу у интернет-оператора {oper1.Name}");

Console.WriteLine(abonent1.PlanRequest(oper1));

Console.WriteLine(oper1);

Console.WriteLine($"Абонент {abonent1.Name} хочет получить услугу у интернет-оператора {oper1.Name}");

Console.WriteLine(abonent1.PlanRequest(oper1));

Console.WriteLine(oper1);

Console.WriteLine($"Абонент {abonent2.Name} хочет получить услугу у интернет-оператора {oper1.Name}");

Console.WriteLine(abonent2.PlanRequest(oper1));

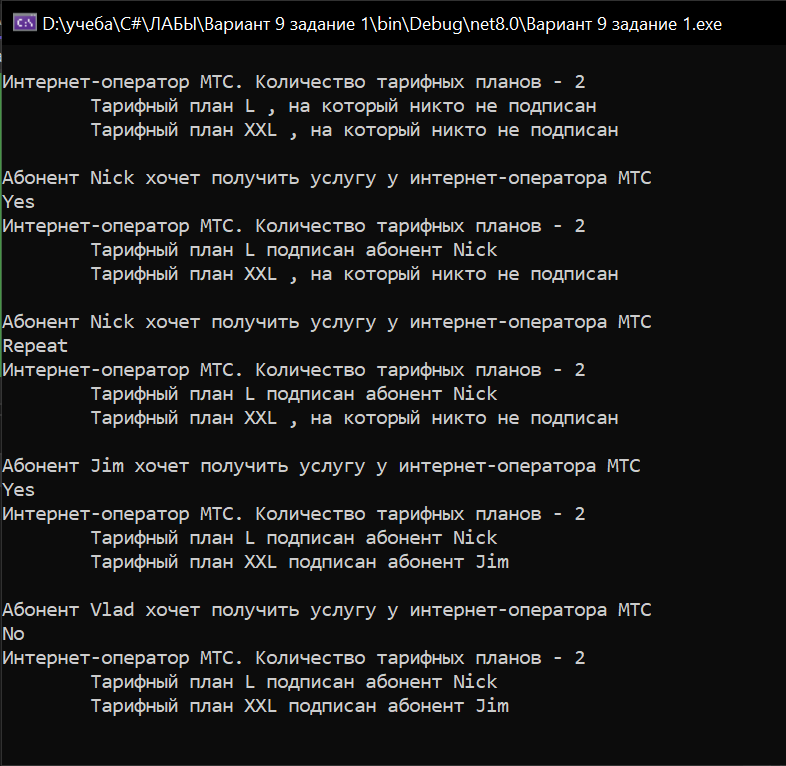
Console.WriteLine(oper1);

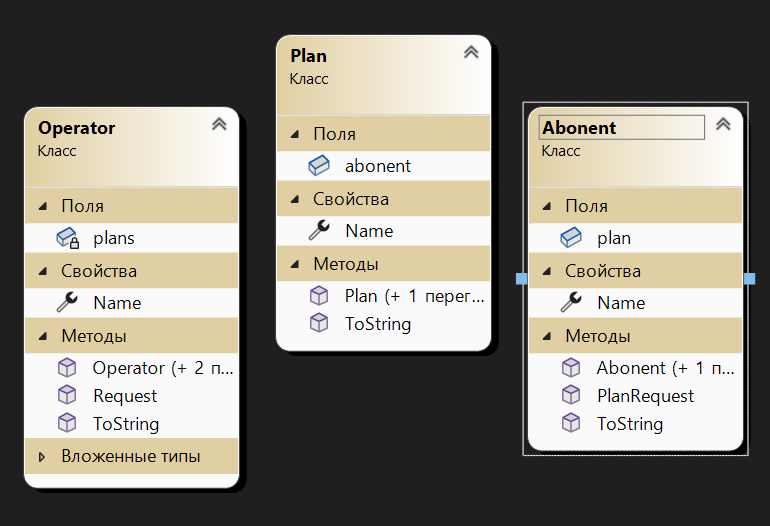
Console.WriteLine($"Абонент {abonent3.Name} хочет получить услугу у интернет-оператора {oper1.Name}");

Console.WriteLine(abonent3.PlanRequest(oper1));

Console.WriteLine(oper1);

Console.ReadKey();





1

M

1

1

Вариант 10

namespace Вариант\_10\_задание\_1

{

internal class Organization

{

public enum Allocation {Yes=1,No=0,Repeat=-1 }

public string Name { get; set; }

Project[] projects=new Project[0];

public Organization():this("Noname")

{

}

public Organization(string name)

{

Name = name;

}

public Allocation Request(Employee employee)

{

foreach(Project p in projects)

{

if(p?.employee?.Name == employee?.Name)

{

return Allocation.Repeat;

}

}

for(int i = 0; i < projects.Length; i++)

{

if (projects[i].employee == null)

{

projects[i].employee = employee;

employee.project= projects[i];

return Allocation.Yes;

}

}

return Allocation.No;

}

public Organization(string name, params Project[] projects) : this(name)

{

this.projects = projects;

}

public override string? ToString()

{

if(projects.Length == 0)

{

return $"Организация не имеет активных проектов";

}

else

{

string str = "Организация ";

if(Name!=null)

{

str += $"{Name}. Кол-во активных проектов - {projects.Length}\n";

}

foreach(Project p in projects)

{

str += $"\t{p.ToString()}\n" ;

}

return str;

}

}

}

}

namespace Вариант\_10\_задание\_1

{

internal class Project

{

public string Name { get; set; }

public Employee employee;

public Project():this("Noname")

{

}

public Project(string name)

{

Name = name;

}

public override string? ToString()

{

string str = "Проект ";

if (Name != null)

{

str += $"{Name} " +

(employee == null ? "никем не выполняется" : $"выполняется сотрудником {employee.Name}");

}

return str;

}

}

}

namespace Вариант\_10\_задание\_1

{

internal class Employee

{

public string Name { get; set; }

public Project project;

public Employee():this("Noname")

{

}

public Employee(string name)

{

Name = name;

}

public Organization.Allocation ProjectRequest(Organization organization)

{

return organization.Request(this);

}

public override string? ToString()

{

string str = "Сотрудник ";

if (Name != null)

{

str += $"{Name} " +

(project == null ? "не привязан к проекту" : $"привязан к проекту {project.Name}");

}

return str;

}

}

}

using Вариант\_10\_задание\_1;

Organization organ=new Organization();

Console.WriteLine(organ);

Project project1 = new Project("Rusak");

Project project2 = new Project("Ermak");

Organization organ1 = new Organization("Hub", project1, project2);

Console.WriteLine(organ1);

Employee employee1 = new Employee("Artur");

Employee employee2 = new Employee("Nick");

Employee employee3 = new Employee("Ken");

Console.WriteLine($"Сотрудник {employee1.Name} организации {organ1.Name} будет назначен на проект");

Console.WriteLine(employee1.ProjectRequest(organ1));

Console.WriteLine(organ1);

Console.WriteLine($"Сотрудник {employee1.Name} организации {organ1.Name} будет назначен на проект");

Console.WriteLine(employee1.ProjectRequest(organ1));

Console.WriteLine(organ1);

Console.WriteLine($"Сотрудник {employee2.Name} организации {organ1.Name} будет назначен на проект");

Console.WriteLine(employee2.ProjectRequest(organ1));

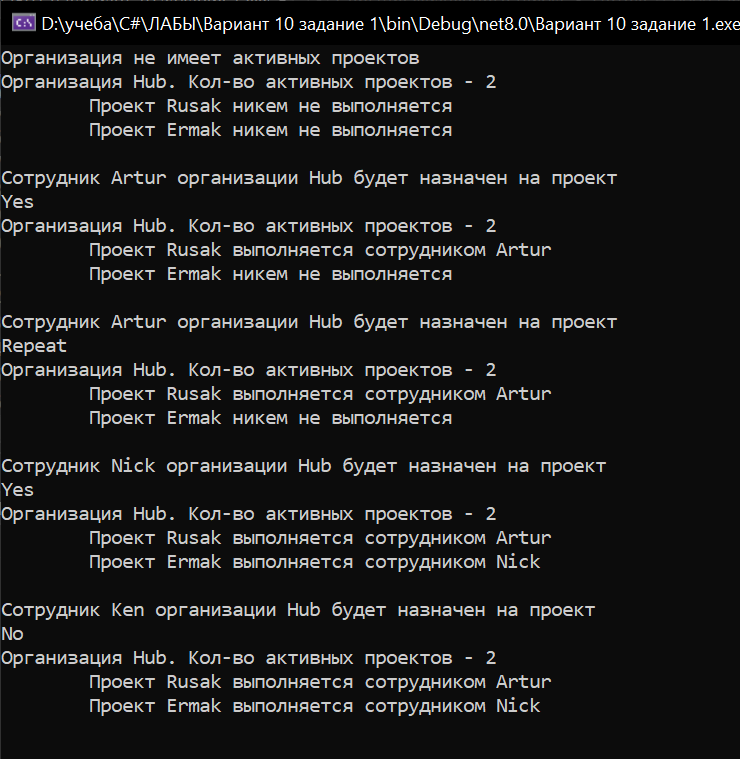
Console.WriteLine(organ1);

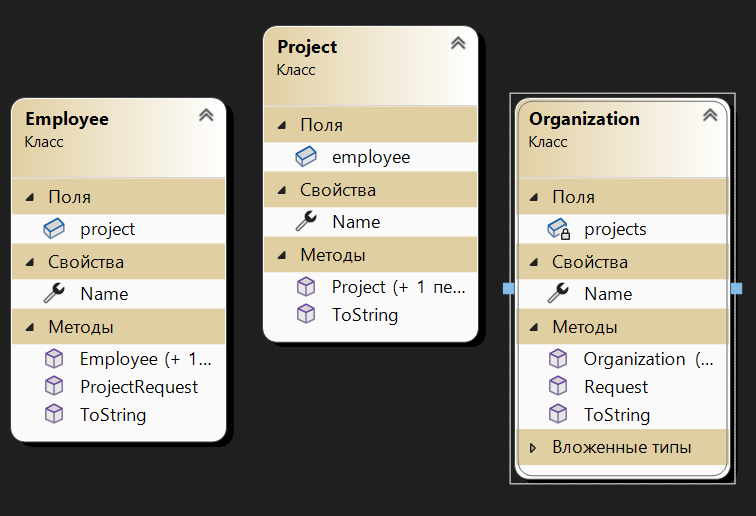
Console.WriteLine($"Сотрудник {employee3.Name} организации {organ1.Name} будет назначен на проект");

Console.WriteLine(employee3.ProjectRequest(organ1));

Console.WriteLine(organ1);

Console.ReadKey();





1

1

1

M

Задание 2

Вариант 2 Здания

using Вариант\_2\_задание\_2\_здания;

baseBuilding baseBuilding = new baseBuilding();

baseBuilding.Print();

Console.WriteLine();

Building baseBuilding1 = new baseBuilding(10,8,"ул.Луговая, 67");

baseBuilding1.Print();

Console.WriteLine($"Количество этажей - {baseBuilding1.KolFloor()}");

Console.WriteLine($"Общая площадь здания - {baseBuilding1.SquareBase()}");

Console.WriteLine();

Building house = new House("ул.Лучная, 56");

house.Print();

Building house1 = new House(6, 10, 10, "ул.Песочная, 32");

house1.Print();

Console.WriteLine($"Количество этажей - {house1.KolFloor()}");

Console.WriteLine($"Общая площадь здания - {house1.SquareBase()}");

Console.WriteLine();

house1.address = "ул.Парковая, 56";

house1.Print();

Console.WriteLine($"Количество этажей - {house1.KolFloor()}");

Console.WriteLine($"Общая площадь здания - {house1.SquareBase()}");

Console.WriteLine();

Console.ReadKey();

namespace Вариант\_2\_задание\_2\_здания

{

abstract class Building

{

public string address;

protected Building(string address)

{

Address = address;

}

protected Building()

{

}

public string Address { get => address; set => address = value; }

public abstract int KolFloor();

public abstract int SquareBase();

public virtual void Print()

{

Console.WriteLine($"Здание по адресу {address}");

}

}

}

namespace Вариант\_2\_задание\_2\_здания

{

internal class baseBuilding:Building

{

private int width;

private int depth;

public baseBuilding(string address):base(address)

{

}

public baseBuilding(int width, int depth, string address):base(address)

{

Width = width;

Depth = depth;

}

public baseBuilding(int width, int depth):base()

{

Width = width;

Depth = depth;

}

public baseBuilding():base()

{

}

public int Width {

get => width;

set

{

if(value>0)

width = value;

}

}

public int Depth {

get => depth;

set

{

if (value>0)

depth = value;

}

}

public override int KolFloor()

{

return 0;

}

public override int SquareBase()

{

return width \* depth;

}

new public void Print()

{

base.Print();

Console.WriteLine($"Фундамент по адресу {address}");

}

}

}

namespace Вариант\_2\_задание\_2\_здания

{

internal sealed class House:Building

{

private int height;

private int width;

private int depth;

public House(string address):base(address)

{

}

public House(int height, int width, int depth, string address):base(address)

{

Height = height;

Width = width;

Depth = depth;

}

public int Height

{

get => height;

set

{

if(value > 0)

height = value;

}

}

public int Width

{

get => width;

set

{

if(value > 0)

width = value;

}

}

public int Depth

{

get => depth;

set

{

if(value > 0)

depth = value;

}

}

public override int KolFloor()

{

return height/3;

}

public override int SquareBase()

{

return width\*depth\*KolFloor();

}

public override void Print()

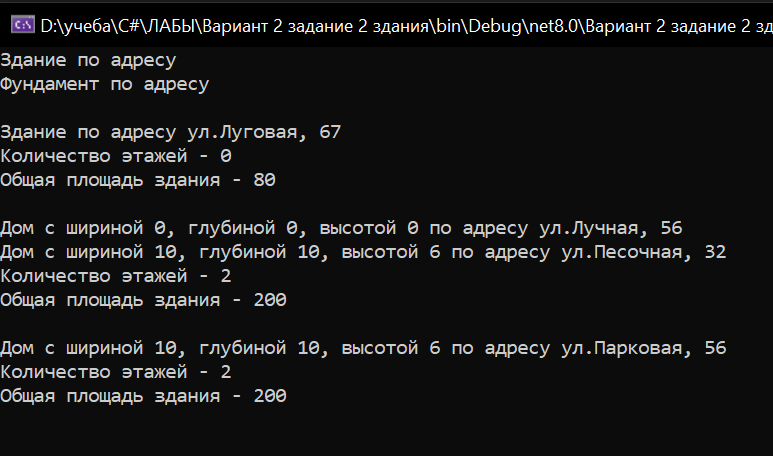
{

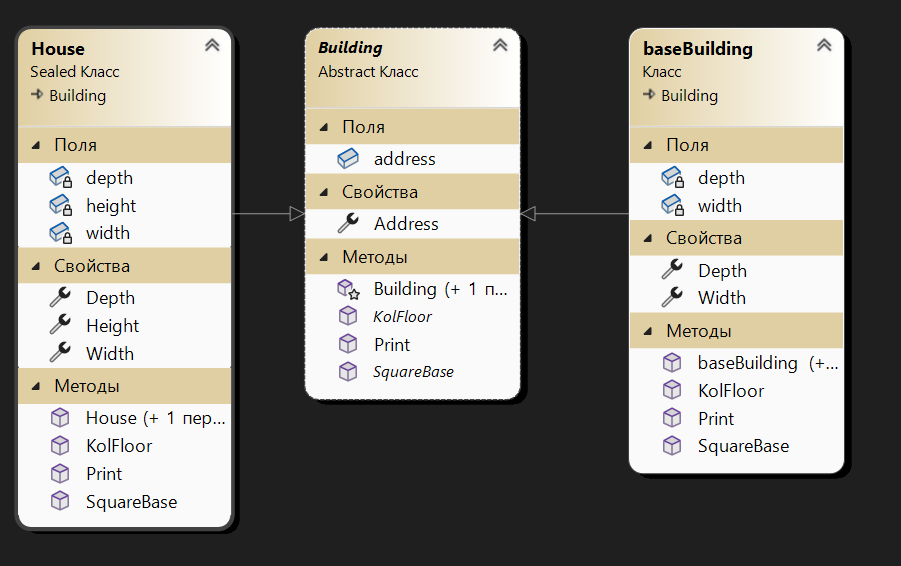
Console.WriteLine($"Дом с шириной {width}, глубиной {depth}, высотой {height} по адресу {address}");

}

}

}





Вариант 3 Транспорт

using вариант\_3\_задание\_2;

Auto auto = new Auto();

auto.Print();

Console.WriteLine();

Transport auto1 = new Auto();

auto1.Print();

Console.WriteLine();

Auto auto2 = new Auto("Audi", 10,60);

auto2.Print();

Console.WriteLine($"Автомобиль {auto2.Marka} проехал {auto2.Distance()} км");

auto2.Stop();

auto2.Print();

Console.WriteLine();

Transport plane = new Plane();

plane.Print();

Plane plane1 = new Plane("Ил-190", 1000, 12, 300);

plane1.Print();

Console.WriteLine($"Самолет {plane1.Name} пролетел {plane1.Distance()} км");

plane1.Stop();

plane1.Print();

Console.WriteLine();

Console.ReadKey();

namespace вариант\_3\_задание\_2

{

abstract class Transport

{

private int speed;

public Transport(int speed)

{

Speed = speed;

}

public Transport()

{

}

public int Speed

{

get => speed;

set

{

if (value >= 0)

speed = value;

}

}

public abstract int Distance();

public abstract void Stop();

public virtual void Print()

{

Console.WriteLine($"Транспорт движется со скоростью {speed}");

}

}

}

namespace вариант\_3\_задание\_2

{

internal class Auto:Transport

{

private string marka;

private int time;

public Auto():base()

{

}

public Auto(string marka, int time, int speed):base(speed)

{

Marka = marka;

Time = time;

}

public string Marka { get => marka; set => marka = value; }

public int Time

{

get => time;

set

{

if(value>0)

{

time = value;

}

}

}

public override int Distance()

{

return time \* Speed;

}

public override void Stop()

{

Speed = 0;

Console.WriteLine("Автомобиль остановился");

return ;

}

new public void Print()

{

base.Print();

Console.WriteLine($"Автомобиль {marka} едет со скоростью {Speed}");

}

}

}

namespace вариант\_3\_задание\_2

{

internal class Plane:Transport

{

private string name;

private int height;

private int time;

public Plane():base()

{

}

public Plane(string name, int height, int time, int speed):base(speed)

{

Name = name;

Height = height;

Time = time;

}

public string Name { get => name; set => name = value; }

public int Height

{

get => height;

set

{

if(value >= 0)

height = value;

}

}

public int Time

{

get => time;

set

{

if(value >= 0)

time = value;

}

}

public override int Distance()

{

return Time \* Speed;

}

public override void Stop()

{

Height = 0;

Console.WriteLine("Самолет приземлился");

return;

}

public override void Print()

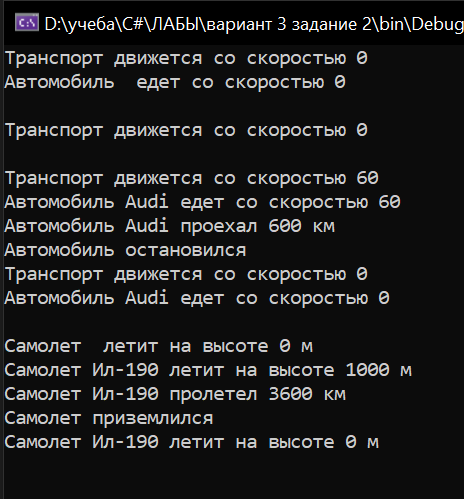
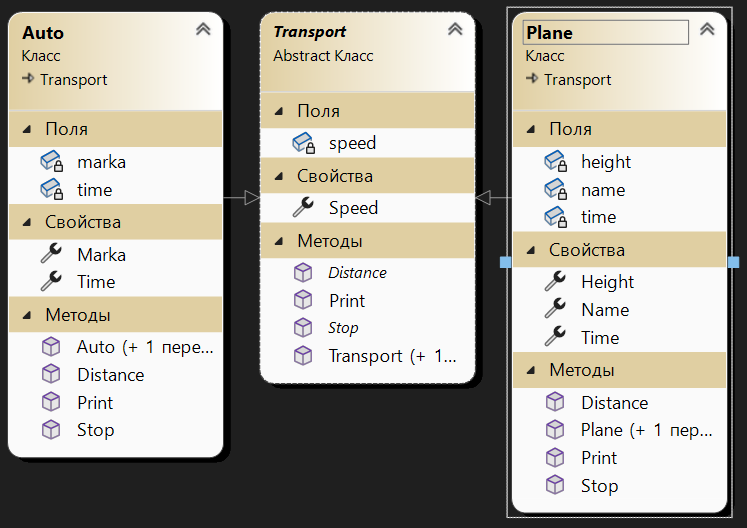
{

Console.WriteLine($"Самолет {Name} летит на высоте {Height} м");

}

}

}



Вариант 5 Товары

using вариант\_5\_задание\_2;

Products product = new Product();

product.Print();

Product product1 = new Product();

product1.Print();

Products toy = new Toy();

toy.Print();

Toy toy1 = new Toy("Мишка","3 этаж, блок В", 200);

toy1.Print();

toy1.ChangePrice(30);

toy1.Print();

toy1.ChangeDepartment("2 этаж, блок Е");

toy1.Print();

Console.ReadKey();

namespace вариант\_5\_задание\_2

{

abstract class Products

{

private double price;

protected Products()

{

}

protected Products(double price)

{

Price = price;

}

public double Price

{

get => price;

set

{

if(value>0)

price = value;

}

}

public abstract void ChangePrice(double change);

public abstract void ChangeDepartment(string department);

public virtual void Print()

{

Console.WriteLine($"Товары");

}

}

}

namespace вариант\_5\_задание\_2

{

internal class Toy:Products

{

private string name;

private string department;

public Toy():base()

{

}

public Toy(string name, string department, double price) : base(price)

{

Name = name;

Department = department;

}

public string Name { get => name; set => name = value; }

public string Department { get => department; set => department = value; }

public override void ChangePrice(double change)

{

Price=Price\*(Math.Abs(100+change)/100);

Console.WriteLine("Цена изменена");

}

public override void ChangeDepartment(string department)

{

Department = department;

Console.WriteLine("Товар перемещен в другой отдел");

}

public override void Print()

{

Console.WriteLine($"Игрушка {Name} находится в отделе {Department} по цене {Price}");

}

}

}

namespace вариант\_5\_задание\_2

{

internal class Product:Products

{

private string nameShop;

public Product():base()

{

}

public Product(string nameShop ,double price) : base(price)

{

NameShop = nameShop;

}

public string NameShop { get => nameShop; set => nameShop = value; }

public override void ChangePrice(double change)

{

return ;

}

public override void ChangeDepartment(string department)

{

return;

}

new public void Print()

{

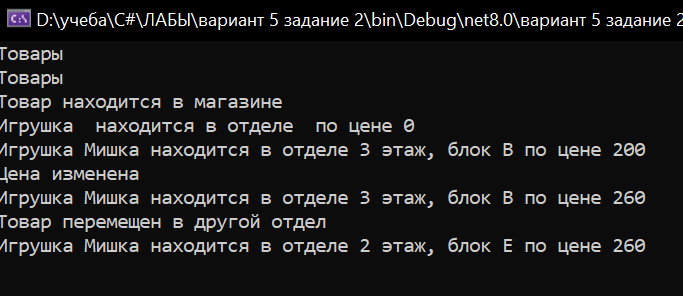
base.Print();

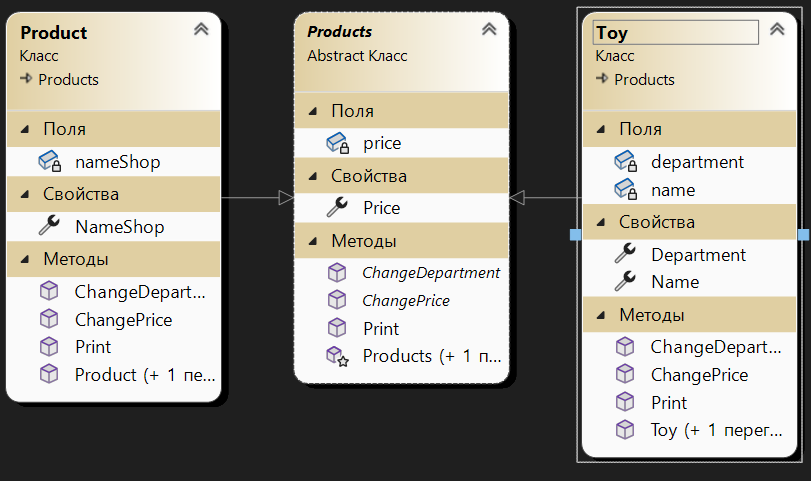
Console.WriteLine($"Товар находится в магазине {NameShop}");

}

}

}





Вариант 10 Налоги

using вариант\_10\_задание\_2;

Taxes taxes = new NDS();

taxes.Print();

Taxes taxes1 = new NDS("John", 3000, 2000);

taxes1.Print();

Console.WriteLine($"Расчет налога для {taxes1.Name} - {taxes1.RaschetTaxe()}");

Console.WriteLine();

NDS nDS = new NDS("Tim", 4500, 3700);

nDS.Print();

nDS.Osvobozhdenie();

nDS.Print();

Podohodny podohodny= new Podohodny("German", 5000, 3500);

Console.WriteLine($"Плательщик {podohodny.Name} обязан уплатить подоходного {podohodny.RaschetTaxe()}");

podohodny.Print();

podohodny.Osvobozhdenie();

podohodny.Print();

Console.ReadKey();

namespace вариант\_10\_задание\_2

{

abstract class Taxes

{

private string name;

private int dohodi;

protected Taxes()

{

}

protected Taxes(string name, int dohodi)

{

Name = name;

Dohodi = dohodi;

}

public string Name { get => name; set => name = value; }

public int Dohodi

{

get => dohodi;

set

{

if(value>=0)

dohodi = value;

}

}

public abstract double RaschetTaxe();

public abstract void Osvobozhdenie();

public virtual void Print()

{

Console.WriteLine($"Плательщик {Name} обязан уплатить налог с налогооблагаемым доходом {Dohodi}");

}

}

}

namespace вариант\_10\_задание\_2

{

internal class NDS:Taxes

{

private int rashodi;

public NDS():base()

{

}

public NDS(string name, int dohodi, int rashodi) : base(name, dohodi)

{

Rashodi = rashodi;

}

public int Rashodi

{

get => rashodi;

set

{

if (value >= 0)

rashodi = value;

}

}

public override double RaschetTaxe()

{

return (Dohodi - Rashodi) / 120 \* 20;

}

public override void Osvobozhdenie()

{

Dohodi = 0;

Rashodi = 0;

Console.WriteLine("Плательщик освобождается от выплаты по льготе");

return;

}

new public void Print()

{

base.Print();

Console.WriteLine($"НДС к уплате для плательщика {Name} - {RaschetTaxe()}");

}

}

}

namespace вариант\_10\_задание\_2

{

internal class Podohodny:Taxes

{

private int vicheti;

public Podohodny():base()

{

}

public Podohodny(string name, int dohodi, int vicheti) : base(name, dohodi)

{

Vicheti = vicheti;

}

public int Vicheti

{

get => vicheti;

set

{

if(value>=0)

vicheti = value;

}

}

public override double RaschetTaxe()

{

return (Dohodi - Vicheti) \* 0.13;

}

public override void Osvobozhdenie()

{

Vicheti = 0;

Dohodi = 0;

Console.WriteLine("Плательщик освобождается от выплаты по льготе");

return;

}

public override void Print()

{

Console.WriteLine($"Плательщик {Name} обязан уплатить подоходного - {RaschetTaxe()}");

}

}

}

