Лабораторная работа 11 по С#

Многопоточное программирование.

студент, заочное, 40331, Василевский Евгений Валерьевич

Задание. Реализовать предложенный вариант задания. Перечисленные объекты по возможности реализовать отдельными классами. Вывод указанных сообщений обязателен. Индивидуальные задания по вариантам:

**Вариант 1**

Вариант 1. Склад 3 производителя поставляют товары на склад. Каждый товар производится определенное к-во времени. Склад имеет ограниченную вместимость (задается в конструкторе). Покупатели забирают случайный товар со склада каждые 2 сек. Когда склад заполнен полностью – он прекращает принимать товары. Программа завершается, когда покупатели раскупят все, что осталось на складе. Сообщения: " => {Fabricator} отгрузил на склад {product}") "{product} куплен. Осталось {balance\_product}({balance\_total})" "Склад закрыт на переучет"

using lab11вар1;

internal class Program

{

static public List<string> fabrNames = new List<string> { "J&J", "DANON", "TESLA" };

static public List<string> prodNames = new List<string> { "мишки","машинки","куклы","миски","вилки","ножи","доски","цемент","обои","рукавицы"};

static public List<int> timeCreat = new List<int> {3, 2, 3, 2, 1 ,1, 3, 2, 2 };

static public List<Product> products = new List<Product>();

private static void Main(string[] args)

{

Console.WriteLine("Лаборраторрная работа 11. Вариант 1. Склад");

Storage storage=new Storage(products,fabrNames,prodNames, timeCreat, 10);

Thread threadStorage = new Thread(storage.addProduct);

threadStorage.Start();

Buyer buyer1 = new Buyer("John", products);

Thread thread1=new Thread(buyer1.Buys);

Buyer buyer2 = new Buyer("Ben", products);

Thread thread2 = new Thread(buyer2.Buys);

Buyer buyer3 = new Buyer("Tony", products);

Thread thread3 = new Thread(buyer3.Buys);

while(products.Count <7)

{

Thread.Sleep(10);

}

thread1.Start();

thread2.Start();

thread3.Start();

Console.ReadKey();

}

}

namespace lab11вар1

{

internal class Product

{

public Product(string name, string fabricator, int timeCreateProduct)

{

Name = name;

Fabricator = fabricator;

TimeCreateProduct = timeCreateProduct;

}

public string Name { get; set; }

public string Fabricator { get; set; }

public int TimeCreateProduct { get; set; }

public override string? ToString()

{

return $"{Name} ";

}

}

}

namespace lab11вар1

{

internal class Storage

{

Random Random = new Random();

List<Product> products;

List<string> fabrNames;

List<string> productNames;

List<int> timesCreate;

public Storage(List<Product> products, List<string> fabrNames, List<string> productNames, List<int> timesCreate, int vmestim)

{

this.products = products;

this.fabrNames = fabrNames;

this.productNames = productNames;

this.timesCreate = timesCreate;

this.vmestim = vmestim;

}

public int vmestim { get; set; }

public void addProduct()

{

while(products.Count < vmestim)

{

Product product = new Product(productNames[Random.Next(0, products.Count)], fabrNames[Random.Next(0, fabrNames.Count)], timesCreate[Random.Next(0,timesCreate.Count)]);

products.Add(product);

Console.WriteLine($"{product.Fabricator} отгрузил на склад {product}");

Thread.Sleep((int)product.TimeCreateProduct\*1000);

}

Console.WriteLine($"Склад закрыт на переучет");

}

}

}

namespace lab11вар1

{

internal class Buyer

{

Random Random = new Random();

List<Product> products;

public Buyer( string name, List<Product> products)

{

this.products = products;

Name = name;

}

public string Name { get; set; }

private bool checkList()

{

return products.Count > 0;

}

public void Buys()

{

while (true)

{

lock (products)

{

if (!checkList()) break;

int index = Random.Next(0, products.Count);

Product product1=products[Random.Next(0, products.Count)];

Console.WriteLine($"{product1} куплен. Осталось {products.Count-1}");

products.RemoveAt(index);

}

Thread.Sleep(9000);

}

Console.WriteLine("Все товары раскуплены");

}

public override string? ToString()

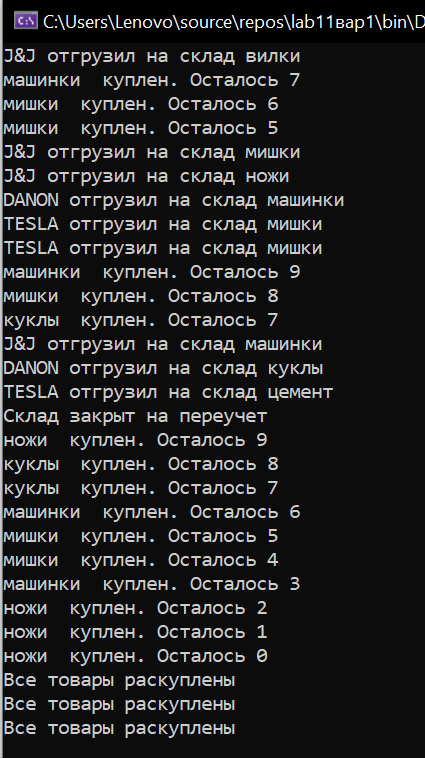
{

return $"{Name} " ;

}

}

}



Вариант 2. Порт

Корабли прибывают в порт каждые 2-5 сек. Каждый корабль везет груз от 1 до 10 тонн. Скорость разгрузки – 1 т/сек. Порт одновременно может принять не более 10 кораблей. Программа завершается, когда в док загружено более 100т. Сообщения: "=> {Ship} с грузом {Cargo}т" " ⇓------ {Ship} покинул порт. В порту {count} кораблей. Общий груз {cargo\_total}" "Порт закрыт"

internal class Port

{

Random random = new Random();

Queue<Ship> ships;

List<string> shipNames;

public int totalCargo;

public int Cargo\_max;

public int Ship\_max;

public object locker = new object();

Semaphore semaphore = new Semaphore(10,10);

public Port(Queue<Ship> ships, List<string> shipNames, int totalCargo, int cargo\_max, int ship\_max)

{

this.ships = ships;

this.shipNames = shipNames;

this.totalCargo = totalCargo;

Cargo\_max = cargo\_max;

Ship\_max = ship\_max;

}

public void addShip()

{

while (totalCargo<Cargo\_max)

{

semaphore.WaitOne();

Ship ship=new Ship(shipNames[random.Next(0,shipNames.Count)], random.Next(1,11));

Console.WriteLine($"=>{ship.Name} с грузом {ship.Cargo} т");

ships.Enqueue(ship);

Thread.Sleep(random.Next(2000,5001));

}

}

public void unloadShip()

{

while (true)

{

Ship currentShip = null;

int shipCount;

lock (locker)

{

if (ships.Count > 0)

{

currentShip = ships.Dequeue();

Thread.Sleep(currentShip.Cargo \* 1000);

totalCargo += currentShip.Cargo;

shipCount = ships.Count;

semaphore.Release();

Console.WriteLine($"<------ {currentShip.Name} покинул порт. В порту {shipCount} кораблей. " +

$"Общий груз {totalCargo}");

if (totalCargo >= Cargo\_max)

{

Console.WriteLine("Порт закрыт");

return;

}

}

}

}

}

public void Run()

{

Thread threadShip1 = new Thread(addShip);

Thread threadShip2 = new Thread(unloadShip);

threadShip1.Start();

threadShip2.Start();

threadShip1.Join();

threadShip2.Join();

}

}

internal class Ship

{

public Ship(string name, int cargo)

{

Name = name;

Cargo = cargo;

}

public string Name { get; set; }

public int Cargo { get; set; }

public override string? ToString()

{

return $"{Name} ";

}

}

using lab11вар2;

internal class Program

{

public static List<string> namesShip = new List<string> {"Maria","Lena","Zemfira","Kolokol","Bravo","Johny","Edmar","Upiter" };

public static Queue<Ship> ships = new Queue<Ship>();

public static int totalCargo = 0;

public static int Cargo\_max = 100;

public static int Ships\_max = 10;

private static void Main(string[] args)

{

Console.WriteLine("Лабораторная работа 11. Вариант 2. Порт");

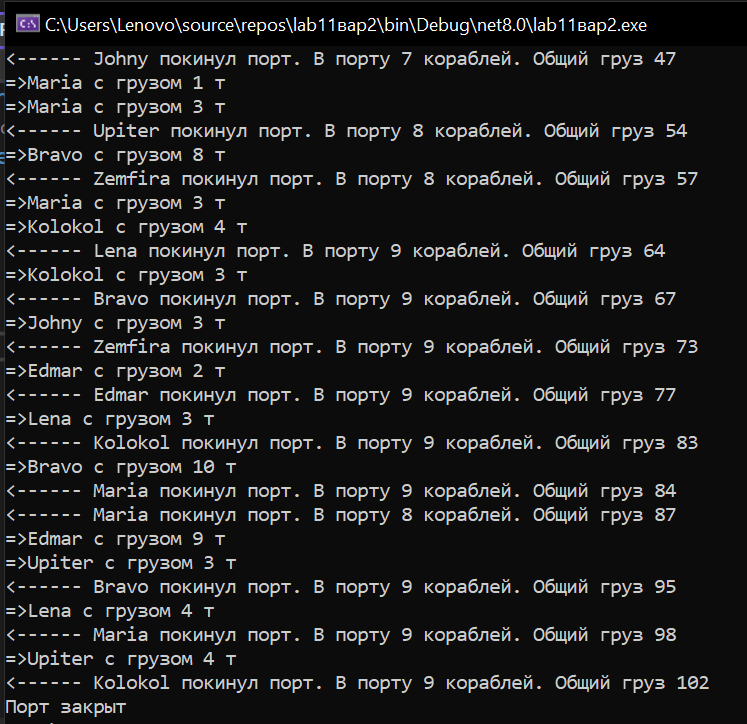
Port port= new Port(ships, namesShip, totalCargo,Cargo\_max,Ships\_max);

port.Run();

Console.ReadKey();

}

}



Вариант 3. Аэропорт

Самолеты прибывают в аэропорт каждые 1-7 сек. У аэропорта 3 взлетные полосы. Между посадкой и взлетом проходит 3 – 5 сек. С вероятностью 10% самолет нуждается в техническом обслуживании и остается в аэропорту. Взлетную полосу он не занимает. Когда в аэропорту более 5 самолетов на ТО, аэропорт закрывается. Сообщения: "{Plane} запрашивает посадку" "=> {Plane} сел на полосу {number}" "{Plane} отправлен на ТО" " ⇓------ {Plane} взлетел . В порту {count} самолетов. На ТО {service} самолетов" "Порт закрыт"

internal class Airport

{

Queue<int> polosi;

Queue<int> polosiPlanes;

Queue<Plane> planes;

List<string> namesPlanes;

Random random=new Random();

public static int Pol\_Max = 3;

public int PlaneTO=0;

public object locker=new object();

Semaphore semaphore = new Semaphore(Pol\_Max,Pol\_Max);

public Airport(Queue<int> polosi, Queue<Plane> planes, List<string> namesPlanes)

{

this.polosi = polosi;

this.planes = planes;

this.namesPlanes = namesPlanes;

this.polosiPlanes = new Queue<int>();

}

public void addPlane()

{

while (true)

{

if (PlaneTO > 5) return;

Thread.Sleep(random.Next(1000,7000));

Plane plane = new Plane(namesPlanes[random.Next(0,namesPlanes.Count)],random.Next(0,10));

Console.WriteLine($"{plane.Name} запрашивает посадку");

semaphore.WaitOne();

lock (locker)

{

planes.Enqueue(plane);

if (polosi.Count > 0)

{

int number = polosi.Dequeue();

polosiPlanes.Enqueue(number);

Console.WriteLine($"=>{plane.Name} сел на полосу {number}");

}

}

}

}

public void Vzlet()

{

while(true)

{

lock (locker)

{

if(planes.Count > 0 && polosiPlanes.Count>0)

{

int ranTO=random.Next(0,10);

Thread.Sleep(random.Next(3000,5000));

Plane plane1=planes.Dequeue();

int pol=polosiPlanes.Dequeue();

if (ranTO == plane1.Remont)

{

PlaneTO++;

Console.WriteLine($"{plane1.Name} отправлен на ТО");

if (PlaneTO > 5)

{

Console.WriteLine("Порт закрыт");

return;

}

}

polosi.Enqueue(pol);

semaphore.Release();

Console.WriteLine($"<------{plane1.Name} взлетел. В порту {planes.Count}. На ТО {PlaneTO} самолетов");

}

else

{

Thread.Sleep(1000);

}

}

}

}

public void Run()

{

Thread thread = new Thread(addPlane);

Thread thread1 = new Thread(Vzlet);

thread.Start();

thread1.Start();

thread1.Join();

thread.Join();

}

}

internal class Plane

{

public string Name { get; set; }

public int Remont;

public Plane(string name, int remont)

{

Name = name;

Remont = remont;

}

public override string? ToString()

{

return $"{Name} ";

}

}

using lab11вар3;

internal class Program

{

public static List<string> planeNames = new List<string> { "Mig","Boing","Kukuruznik","Bat","Tu"};

public static Queue<Plane> planes = new Queue<Plane>();

public static Queue<int> polosi= new Queue<int>();

private static void Main(string[] args)

{

Console.WriteLine("Лабораторная работа 11. Вариант 3. Аэропорт");

polosi.Enqueue(1);

polosi.Enqueue(2);

polosi.Enqueue(3);

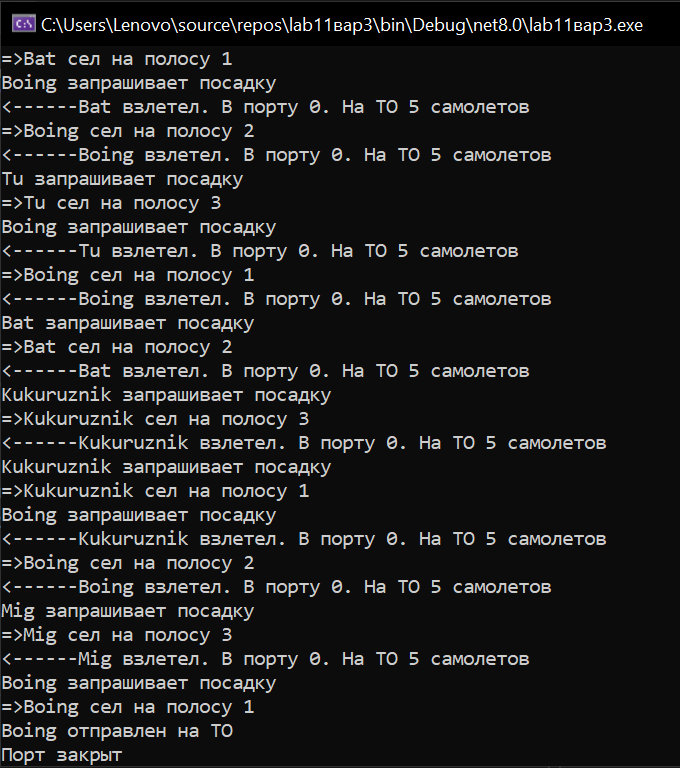
Airport airport = new Airport(polosi, planes, planeNames);

airport.Run();

Console.ReadKey();

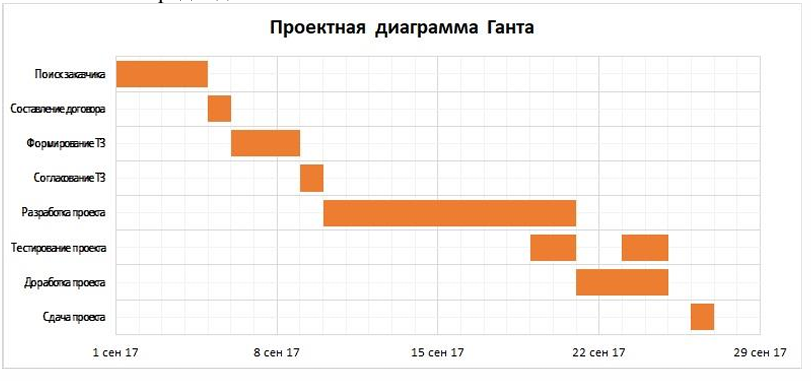
}

}



Вариант 4. Тайм-менеджмент

На основе диаграммы Ганта (можно взять любую из Интернета), эмулировать выполнение различных задач. Каждая задача может либо начинаться по завершении предыдущей, либо заканчиваться вместе с ней. Длительность каждой задачи может увеличиться из-за непредвиденных обстоятельств на 5 – 50%. Сообщения: "{Task}. Время начала{time}" "{Task}. Время завершения{time}" "Проект закончен"



using lab11вар4;

internal class Program

{

public static List<Task1> tasks = new List<Task1>();

private static void Main(string[] args)

{

Console.WriteLine("Лабораторная работа 11. Вариант 4. Тайм-менеджмент");

Task1 poiskZakazchika = new Task1("Поиск заказчика", 5);

Task1 SostavlenieDogovora = new Task1("Составление договора", 2);

Task1 FormirovanieTZ = new Task1("Формирование ТЗ", 4);

Task1 SoglasovanieTZ = new Task1("Согласование ТЗ", 2);

Task1 RazrabotkaProekta = new Task1("Разработка проекта", 10);

Task1 TestirovanieProekta = new Task1("Тестирование проекта", 4);

Task1 DorabotkaProekta = new Task1("Доработка проекта", 5);

Task1 SdachaProekta = new Task1("Сдача проекта", 1);

SostavlenieDogovora.Dependencies.Add(poiskZakazchika);

FormirovanieTZ.Dependencies.Add(SostavlenieDogovora);

SoglasovanieTZ.Dependencies.Add(FormirovanieTZ);

RazrabotkaProekta.Dependencies.Add(SoglasovanieTZ);

TestirovanieProekta.Dependencies.Add(RazrabotkaProekta);

DorabotkaProekta.Dependencies.Add(TestirovanieProekta);

SdachaProekta.Dependencies.Add(DorabotkaProekta);

tasks.Add(poiskZakazchika);

tasks.Add(SostavlenieDogovora);

tasks.Add(FormirovanieTZ);

tasks.Add(SoglasovanieTZ);

tasks.Add(RazrabotkaProekta);

tasks.Add(TestirovanieProekta);

tasks.Add(DorabotkaProekta);

tasks.Add(SdachaProekta);

Project project=new Project(tasks);

project.ExecuteProject();

Console.ReadKey();

}

}

internal class Task1

{

public string Name { get; set; }

public int Duration { get; set; }

public DateTime StartTime { get; set; }

public DateTime EndTime { get; set; }

public List<Task1> Dependencies { get; set; }

public bool IsCompleted { get; set; }

public Thread WorkThread { get; set; }

public Task1(string name, int duration)

{

Name = name;

Duration = duration;

Dependencies= new List<Task1>();

IsCompleted = false;

}

}

internal class Project

{

List<Task1> tasks;

public object locker=new object();

DateTime projectStart = new DateTime(2017, 9, 1);

Semaphore semaphore = new Semaphore(1, 1);

Random rand = new Random();

public Project(List<Task1> tasks)

{

this.tasks = tasks;

}

public void ExecuteTask(Task1 task, DateTime projectStart)

{

foreach (var dependency in task.Dependencies)

{

while (!dependency.IsCompleted)

{

Thread.Sleep(1000);

}

}

semaphore.WaitOne();

lock (locker)

{

// Находим самое позднее время завершения зависимостей

DateTime startTime = projectStart;

foreach (var dependency in task.Dependencies)

{

if (dependency.EndTime > startTime)

startTime = dependency.EndTime;

}

task.StartTime = startTime;

task.EndTime = startTime.AddDays(task.Duration);

Console.WriteLine($"{task.Name}. Время начала {task.StartTime.ToShortDateString()}");

}

Thread.Sleep(task.Duration \* 1000\*(100+rand.Next(5,50))/100);

semaphore.Release();

lock (locker)

{

task.IsCompleted = true;

Console.WriteLine($"{task.Name}. Время завершения {task.EndTime.ToShortDateString()}");

}

}

public void ExecuteProject()

{

foreach (var task in tasks)

{

task.WorkThread = new Thread(() =>

{

ExecuteTask(task, projectStart);

});

}

foreach (var task in tasks)

{

task.WorkThread.Start();

}

foreach (var task in tasks)

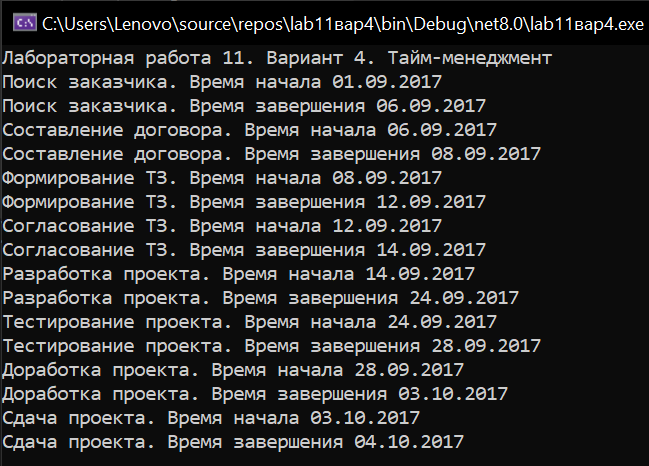
{

task.WorkThread.Join();

}

}

}



Вариант 5. Библиотека

В библиотеке 3 читальных зала. Каждый вмещает определенное количество читателей (задается в конструкторе). Читатели заходят в библиотеку каждые 2 сек. Зал выбирается случайным образом. Читатель проводит в библиотеке от 2 до 5 сек. Библиотека работает 10 сек. Сообщения: "=>Читатель {number} прошел в зал {reading\_room}. Всего читателей {count}(count\_total)" "Читатель {number} покинул библиотеку" "Библиотека закрыта""

using lab11вар5;

internal class Program

{

public static Queue<int> ints = new Queue<int>();

public static List<Queue<Reader>> readers = new List<Queue<Reader>>

{

new Queue<Reader>(),

new Queue<Reader>(),

new Queue<Reader>()

};

private static void Main(string[] args)

{

Console.WriteLine("Лабораторная работа 11. Вариант 5. Библиотека");

ints.Enqueue(1);

ints.Enqueue(2);

ints.Enqueue(3);

ints.Enqueue(4);

ints.Enqueue(5);

ints.Enqueue(6);

ints.Enqueue(7);

ints.Enqueue(8);

ints.Enqueue(9);

ints.Enqueue(10);

ReadingRoom readingRoom1=new ReadingRoom(1,2);

ReadingRoom readingRoom2=new ReadingRoom(2,1);

ReadingRoom readingRoom3=new ReadingRoom(3,3);

Library library=new Library(readingRoom1,readingRoom2,readingRoom3,ints,readers);

library.Work();

Console.ReadKey();

}

}

internal class Library

{

Random rand = new Random();

List<ReadingRoom> roomList = new List<ReadingRoom>();

ReadingRoom room1;

ReadingRoom room2;

ReadingRoom room3;

Queue<int> NumberReaders;

List<Queue<Reader>> ReaderList;

public int c = 0;

public int Reader\_Count = 0;

public object locker=new object();

List<Semaphore> semaphoreList = new List<Semaphore>();

public Library(ReadingRoom room1, ReadingRoom room2, ReadingRoom room3, Queue<int> numberReaders, List<Queue<Reader>> readers)

{

this.room1 = room1;

this.room2 = room2;

this.room3 = room3;

ReaderList = readers;

NumberReaders = numberReaders;

semaphoreList = new List<Semaphore>

{

new Semaphore(room1.KolReaders, room1.KolReaders),

new Semaphore(room2.KolReaders,room2.KolReaders),

new Semaphore(room3.KolReaders,room3.KolReaders)

};

roomList = new List<ReadingRoom>{room1,room2,room3 };

}

public void addReader()

{

while (Reader\_Count<=6)

{

lock (locker)

{

Reader reader=new Reader(NumberReaders.Dequeue());

int NumberZala = rand.Next(0, 3);

semaphoreList[NumberZala].WaitOne();

Reader\_Count++;

c++;

ReaderList[NumberZala].Enqueue(reader);

Console.WriteLine($"=>Читатель {reader.Number} прошел в зал {roomList[NumberZala]}. Всего читателей {Reader\_Count}");

Thread.Sleep(2000);

}

if (c > 6)

{

Console.WriteLine("Библиотека закрыта");

return;

}

}

}

public void endReader()

{

while (true)

{

lock (locker)

{

if (Reader\_Count > 0)

{

Thread.Sleep(rand.Next(2000, 5000));

int NumberZala=rand.Next(0, 3);

if (ReaderList[NumberZala].Count > 0)

{

Reader reader1 = ReaderList[NumberZala].Dequeue();

Reader\_Count--;

Console.WriteLine($"Читатель {reader1.Number} покинул библиотеку");

semaphoreList[NumberZala].Release();

}

}

}

}

}

public void Work()

{

Thread thread = new Thread(addReader);

Thread thread2 = new Thread(endReader);

thread.Start();

thread2.Start();

thread.Join();

thread2.Join();

}

}

internal class Reader

{

public Reader(int number)

{

Number = number;

}

public int Number { get; set; }

public override string? ToString()

{

return $"Читатель {Number} ";

}

}

internal class ReadingRoom

{

public ReadingRoom(int nomer, int kolReaders)

{

Nomer = nomer;

KolReaders = kolReaders;

}

public int Nomer { get; set; }

public int KolReaders { get; set; }

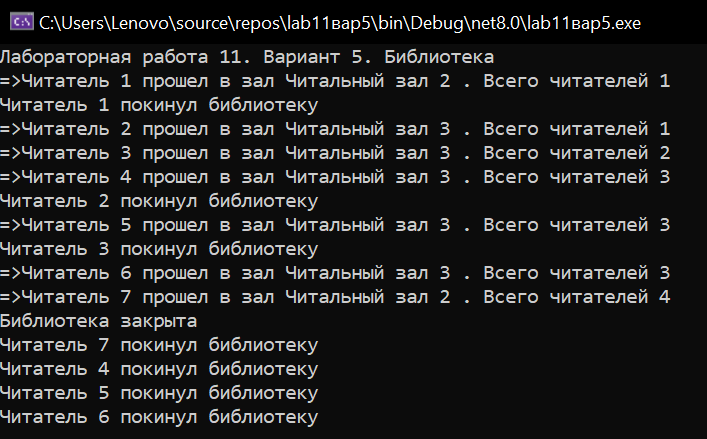
public override string? ToString()

{

return $"Читальный зал {Nomer} ";

}

}



Вариант 6. Стрельба по тарелочкам

Автомат выпускает в воздух тарелочки каждую секунду (тарелочки пронумерованы). 5 стрелков стреляют по тарелочкам. Тарелочка летит в определенный сектор (5 вариантов). Каждый игрок выбирает тарелочку, по которой он стреляет, случайным образом. Если тарелочка в его секторе, он сбивает ее 100%, если в чужом – с вероятностью 30%. Сбитая тарелочка исчезает. Игрок, который сбил 10 тарелочек, заканчивает стрельбу. "Тарелка № {Number} летит в сектор {Sector}" "Стрелок {Name} сбил тарелку № {plate.Number} {hit}:{pass}" "Стрелок {Name} промазал по тарелке № {plate.Number} {hit}:{pass}" "Стрелок {Name} стрельбу закончил" "Игра окончена"

using lab11вар6;

internal class Program

{

public static Queue<Plate> plates = new Queue<Plate>();

public static List<Shooter> shooters = new List<Shooter>

{

new Shooter("Ben", 1),

new Shooter("Don", 2),

new Shooter("Edward", 3),

new Shooter("Kolin", 4),

new Shooter("Denis", 5)

};

private static void Main(string[] args)

{

Console.WriteLine("Лабораторная работа 11. Вариант 6. Стрельба по тарелочкам");

Random random = new Random();

for (int i = 1; i < 200; i++)

{

int ran=random.Next(1,6);

plates.Enqueue(new Plate(i,ran));

}

Game game=new Game(plates, shooters);

game.GameRun();

Console.ReadKey();

}

}

internal class Plate

{

public Plate(int number, int sector)

{

Number = number;

Sector = sector;

}

public int Number { get; set; }

public int Sector { get; set; }

public override string? ToString()

{

return $"{Number} ";

}

}

internal class Shooter

{

public Shooter(string name, int sector)

{

Name = name;

Sector = sector;

Hit = 0;

Pass = 0;

}

public string Name { get; set; }

public int Sector { get; set; }

public int Hit { get; set; }

public int Pass { get; set; }

public override string? ToString()

{

return $"{Name} ";

}

}

internal class Game

{

Random random = new Random();

public Queue<Plate> Plates;

public List<Shooter> Shooters;

public object locker = new object();

public Game(Queue<Plate> plates, List<Shooter> shooters)

{

Plates = plates;

Shooters = shooters;

}

public void Init()

{

while (true)

{

for (int i = 0; i < Shooters.Count; i++)

{

if (Shooters[i].Hit == 10) return;

}

lock (locker)

{

Plate plate=Plates.Peek();

Thread.Sleep(1000);

Console.WriteLine($"Тарелка №{plate.Number} летит в сектор {plate.Sector}");

}

}

}

public void Shoot()

{

while (true)

{

lock (locker)

{

Plate plate = Plates.Peek();

Thread.Sleep(1000);

int ran=random.Next(0,5);

if (plate.Sector-1 == ran)

{

Plate plate1 = Plates.Dequeue();

Shooters[ran].Hit++;

Console.WriteLine($"Стрелок {Shooters[ran].Name} сбил тарелку № {plate1.Number} {Shooters[ran].Hit}:{Shooters[ran].Pass}");

}

else

{

int a = 1;

int ver=random.Next(0,3);

if (a == ver)

{

Plate plate1 = Plates.Dequeue();

Shooters[ran].Hit++;

Console.WriteLine($"Стрелок {Shooters[ran].Name} сбил тарелку № {plate1.Number} {Shooters[ran].Hit}:{Shooters[ran].Pass}");

}

else

{

Plate plate1 = Plates.Dequeue();

Shooters[ran].Pass++;

Console.WriteLine($"Стрелок {Shooters[ran].Name} промазал по тарелке № {plate1.Number} {Shooters[ran].Hit}:{Shooters[ran].Pass}");

}

}

if (Shooters[ran].Hit == 10)

{

Console.WriteLine($"Стрелок {Shooters[ran].Name} стрельбу закончил");

Console.WriteLine("Игра окончена");

return;

}

}

}

}

public void GameRun()

{

Thread thread = new Thread(Init);

Thread thread1 = new Thread(Shoot);

thread.Start();

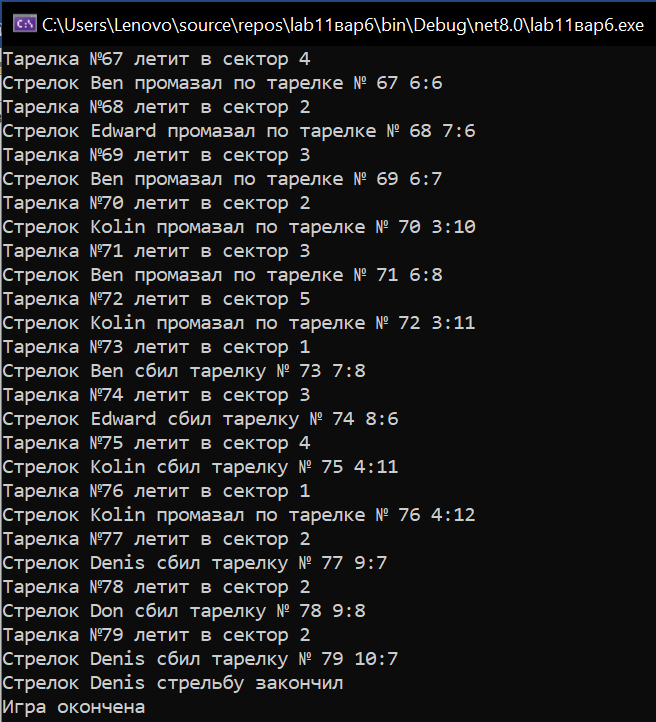
thread1.Start();

thread.Join();

thread1.Join();

}

}



Вариант 8. Лифт

В доме 9 этажей и 3 лифта. На различных этажах случайным образом появляются пассажиры. Каждому пассажиру требуется переместиться на другой этаж (также выбирается случайным образом). По желанию – можно повысить вероятность первого этажа в качестве отправной или конечной точки маршрута пассажиров. Каждый лифт должен быть представлен отдельным потоком. Запросы на вызов лифта должны добавляться в общую очередь. Лифт должен перемещаться к этажу вызова, открывать двери и затем двигаться к этажу назначения. Время движения лифта между этажами фиксировано (например, 2 секунды на этаж). После завершения обслуживания запроса, лифт должен возвращаться в состояние ожидания, готовый к следующему вызову. После выполнения 10 рейсов лифт прекращает работу. Сообщения: "{passenger.Name} едет с этажа {passenger.floorIn} на этаж {passenger.floorOut} " "{lift.Number} едет с этажа {floor1} на этаж {floor2}" "{lift.Number} закончил работу"

using lab11вар8;

internal class Program

{

public static Queue<Passenger> passengers=new Queue<Passenger>();

public static List<string> passNames=new List<string> { "Ben","Tom","Bruce","Steve","Mike","John","Tony","Edmar","Pip"};

private static void Main(string[] args)

{

Console.WriteLine("Лабораторная работа 11. Вариант 8. Лифт");

Lift lift1=new Lift(1, passengers);

Lift lift2=new Lift(2, passengers);

Lift lift3=new Lift(3, passengers);

Work work=new Work(passengers,passNames, lift1,lift2,lift3);

Thread WorkThread =new Thread(work.StartWork);

WorkThread.Start();

while(passengers.Count <5)

{

Thread.Sleep(100);

}

Thread thread1=new Thread(lift1.VizovLift);

Thread thread2=new Thread(lift2.VizovLift);

Thread thread3=new Thread(lift3.VizovLift);

thread1.Start();

thread2.Start();

thread3.Start();

thread1.Join();

thread2.Join();

thread3.Join();

WorkThread.Join();

Console.ReadKey();

}

}

internal class Lift

{

public int Number { get; set; }

public int kolPoez { get; set; }

Queue<Passenger> passengers { get; set; }

object locker = new object();

public Lift(int number, Queue<Passenger> passengers)

{

Number = number;

this.passengers = passengers;

}

public void VizovLift()

{

while (true)

{

lock (locker)

{

if(passengers.Count > 0)

{

Passenger p = passengers.Dequeue();

Console.WriteLine($"{Number} едет с этажа {p.FloorIn} на этаж {p.FloorOut}");

kolPoez++;

Thread.Sleep(2000\*Math.Abs(p.FloorIn-p.FloorOut));

}

}

if(kolPoez == 10)

{

Console.WriteLine($"{Number} закончил работу");

return;

}

}

}

public override string? ToString()

{

return $"{Number} ";

}

}

internal class Passenger

{

public Passenger(string name, int floorIn, int floorOut)

{

Name = name;

FloorIn = floorIn;

FloorOut = floorOut;

}

public string Name { get; set; }

public int FloorIn { get; set; }

public int FloorOut { get; set; }

public override string? ToString()

{

return $"{Name} ";

}

}

internal class Work

{

Random random = new Random();

Random random1 = new Random();

Queue<Passenger> passengers;

List<String> passesNames;

Lift lift1;

Lift lift2;

Lift lift3;

object locker = new object();

public Work(Queue<Passenger> passengers, List<string> passesNames, Lift lift1, Lift lift2, Lift lift3)

{

this.passengers = passengers;

this.passesNames = passesNames;

this.lift1 = lift1;

this.lift2 = lift2;

this.lift3 = lift3;

}

public void StartWork()

{

while (true)

{

lock (locker)

{

Passenger passenger = new Passenger(passesNames[random.Next(0,passesNames.Count)],random.Next(1,10),random1.Next(1,10));

Thread.Sleep(1000);

passengers.Enqueue(passenger);

Console.WriteLine($"{passenger.Name} едет с этажа {passenger.FloorIn} на этаж {passenger.FloorOut}");

}

if(lift1.kolPoez == 10 && lift2.kolPoez == 10 && lift3.kolPoez == 10)

{

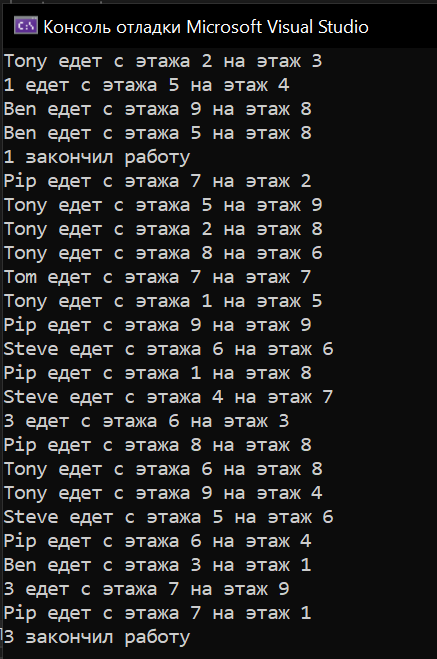
return;

}

}

}

}



Вариант 9. Курьерская доставка

В системе доставки работают 5 курьеров. У каждого курьера свой вид транспорта с определенной скоростью. Заказы поступают из общей очереди. Для каждого заказа есть расстояние, на которое необходимо совершить доставку. После каждой доставки курьер возвращается в офис. Время доставки зависит от скорости курьера и расстояния. Если очередь содержит больше 10 клиентов, офис перестает добавлять клиентов в очередь. Курьеры развозят оставшиеся заказы в очереди и уходят домой. Сообщения: "{klient.Name} требует доставку. Расстояние {klient.Distance}" "{courier.Name} принял заказ {klient.Name}. Время доставки {time} " "{courier.Name } работу окончил." "Офис закрыт"

using lab11вар9;

internal class Program

{

public static Queue<Client> clients = new Queue<Client>();

public static List<string> clientsNames = new List<string> { "Ben", "Tom", "Bruce", "Steve", "Mike", "John", "Tony", "Edmar", "Pip" };

private static void Main(string[] args)

{

Console.WriteLine("Лабораторная работа 11. Вариант 9. Курьерская доставка");

Office office = new Office(clients, clientsNames);

Thread threadOffice = new Thread(office.StartWork);

threadOffice.Start();

Courier courier1 = new Courier("Дима",clients, 5);

Courier courier2 = new Courier("Витя", clients, 4);

Courier courier3 = new Courier("Никита", clients, 3);

Courier courier4 = new Courier("Саша", clients, 6);

Courier courier5 = new Courier("Артем", clients, 2);

Thread thread1 = new Thread(courier1.Dostavka);

Thread thread2 = new Thread(courier2.Dostavka);

Thread thread3 = new Thread(courier3.Dostavka);

Thread thread4 = new Thread(courier4.Dostavka);

Thread thread5 = new Thread(courier5.Dostavka);

while(clients.Count<9)

{

Thread.Sleep(1000);

}

thread1.Start();

thread2.Start();

thread3.Start();

thread4.Start();

thread5.Start();

thread1.Join();

thread2.Join();

thread3.Join();

thread4.Join();

thread5.Join();

threadOffice.Join();

Console.ReadKey();

}

}

internal class Client

{

public Client(string name, double distance)

{

Name = name;

Distance = distance;

}

public string Name { get; set; }

public double Distance { get; set; }

public override string? ToString()

{

return $"{Name} ";

}

}

internal class Courier

{

public string Name { get; set; }

Queue<Client> clients;

public double Skorost { get; set; }

public object locker = new object();

public Courier(string name, Queue<Client> clients, double skorost)

{

Name = name;

this.clients = clients;

Skorost = skorost;

}

public void Dostavka()

{

while (true)

{

lock (locker)

{

Client client = null;

if(clients.Count > 0)

{

client = clients.Dequeue();

if (client != null)

{

double time = client.Distance / Skorost;

Console.WriteLine($"Курьер {Name} принял заказ {client.Name}. Время доставки {time:0.00}");

Thread.Sleep(1000 \* (int)time \* 2);

}

}

}

if (clients.Count == 0)

{

Console.WriteLine($"Курьер {Name} работу окончил.");

return;

}

}

}

public override string? ToString()

{

return $"{Name} ";

}

}

internal class Office

{

Random random = new Random();

Queue<Client> clients;

List<string> clientsNames;

object locker = new object();

public Office(Queue<Client> clients, List<string> clientsNames)

{

this.clients = clients;

this.clientsNames = clientsNames;

}

public void StartWork()

{

while (true)

{

lock (locker)

{

Client client=new Client(clientsNames[random.Next(0,clientsNames.Count)], random.Next(1,40));

clients.Enqueue(client);

Thread.Sleep(1000);

Console.WriteLine($"{client.Name} требует доставку. Расстояние {client.Distance}");

}

if(clients.Count ==10)

{

Console.WriteLine("Офис закрыт");

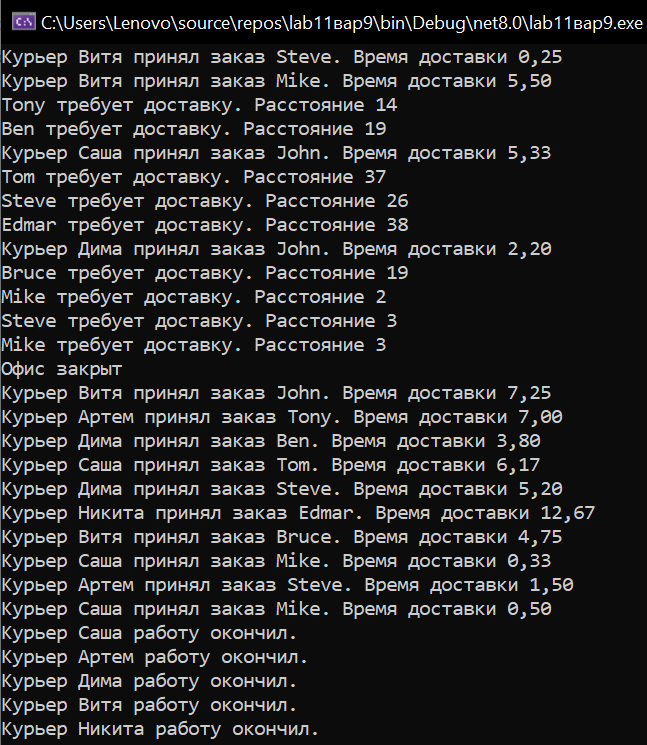
return;

}

}

}

}



Вариант 10. Банк

В банке работают 5 клерков. Клиенты поступают из общей очереди. Каждому клиенту требуется совершить определенную операцию (вклад, снятие, перевод). Каждая операция требует определенного времени. Если очередь содержит больше 10 клиентов, банк перестает добавлять клиентов в очередь. Клерки обслуживают оставшихся в очереди и уходят домой. Сообщения: "{klient.Name} требуется совершить операцию {klient.operation}" "{klerk.Name} принял {klient.Name}. Время операции {time} " "{klerk.Name } работу окончил." "Банк закрыт"

using lab11вар10;

internal class Program

{

public static Queue<Client> clients = new Queue<Client>();

public static List<string> clientsNames = new List<string> { "Ben", "Tom", "Bruce", "Steve", "Mike", "John", "Tony", "Edmar", "Pip" };

public static List<int> Operations = new List<int> {7,4,9 };

private static void Main(string[] args)

{

Console.WriteLine("Лабораторная работа 11. Вариант 10. Банк");

Bank bank=new Bank(clients,clientsNames,Operations);

Thread threadBank = new Thread(bank.StartWork);

threadBank.Start();

while (clients.Count < 8)

{

Thread.Sleep(1000);

}

Klerk klerk1 = new Klerk("Витя", clients);

Klerk klerk2 = new Klerk("Дима", clients);

Klerk klerk3 = new Klerk("Вася", clients);

Klerk klerk4 = new Klerk("Андрей", clients);

Klerk klerk5 = new Klerk("Константин", clients);

Thread thread1=new Thread(klerk1.Obslushivanie);

Thread thread2 = new Thread(klerk2.Obslushivanie);

Thread thread3 = new Thread(klerk3.Obslushivanie);

Thread thread4 = new Thread(klerk4.Obslushivanie);

Thread thread5 = new Thread(klerk5.Obslushivanie);

thread1.Start();

thread2.Start();

thread3.Start();

thread4.Start();

thread5.Start();

thread1.Join();

thread2.Join();

thread3.Join();

thread4.Join();

thread5.Join();

threadBank.Join();

Console.ReadKey();

}

}

internal class Client

{

Random random = new Random();

public Client(string name, int operClient)

{

Name = name;

OperClient = operClient;

}

public string Name { get; set; }

public int OperClient { get; set; }

public override string? ToString()

{

return $"{Name} ";

}

}

internal class Klerk

{

public string Name { get; set; }

Queue<Client> clients;

public object locker = new object();

public Klerk(string name, Queue<Client> clients)

{

Name = name;

this.clients = clients;

}

public void Obslushivanie()

{

while (true)

{

Client client = null;

lock (locker)

{

if(clients.Count > 0)

{

client = clients.Dequeue();

if(client != null )

{

Console.WriteLine($"Клерк {Name} принял {client.Name}. Время операции {client.OperClient} ");

Thread.Sleep(1000\*client.OperClient);

}

}

}

if( clients.Count==0 )

{

Console.WriteLine($"Клерк {Name} работу окончил.");

return;

}

}

}

}

internal class Bank

{

public enum OPERATION {Vklad =5, Snytie=6, Perevod=7 };

Random random = new Random();

public Queue<Client> clients;

List<string> namesClients;

List<int> operClients;

object locker = new object();

public Bank(Queue<Client> clients, List<string> namesClients, List<int> operClients)

{

this.clients = clients;

this.namesClients = namesClients;

this.operClients = operClients;

}

public void StartWork()

{

while (true)

{

lock (locker)

{

Client client = new Client(namesClients[random.Next(0,namesClients.Count)], random.Next(5,8));

clients.Enqueue(client);

Console.WriteLine($"{client.Name} требуется совершить операцию {(OPERATION)client.OperClient}");

Thread.Sleep(1000);

}

if (clients.Count == 10)

{

Console.WriteLine("Банк закрыт");

return;

}

}

}

}

