Создать класс на языке C#, который:

- называется TaskQueue и реализует логику пула потоков;

- создает указанное количество потоков пула в конструкторе;

- содержит очередь задач в виде делегатов без параметров:

delegate void TaskDelegate();

- обеспечивает постановку в очередь и последующее выполнение делегатов с помощью метода

void EnqueueTask(TaskDelegate task).

**public class TaskQueue**

**{**

**private List<Thread> threads;**

**private Queue<Action> tasks;**

**public TaskQueue(int threadCount)**

**{**

**tasks = new Queue<Action>();**

**threads = new List<Thread>();**

**for (int i = 0; i < threadCount; i++)**

**{**

**var t = new Thread(DoThreadWork);**

**threads.Add(t);**

**t.IsBackground = true;**

**t.Start();**

**}**

**}**

**public void EnqueueTask(Action task)**

**{**

**lock (tasks)**

**{**

**tasks.Enqueue(task);**

**Monitor.Pulse(tasks);**

**}**

**}**

**private Action DequeueTask()**

**{**

**lock (tasks)**

**{**

**while (tasks.Count == 0)**

**Monitor.Wait(tasks);**

**return tasks.Dequeue();**

**}**

**}**

**private void DoThreadWork()**

**{**

**while (true)**

**{**

**Action task = DequeueTask();**

**try**

**{**

**task();**

**}**

**catch (ThreadAbortException)**

**{**

**Thread.ResetAbort();**

**}**

**catch (Exception ex)**

**{**

**Console.WriteLine(ex);**

**}**

**}**

**}**

Создать класс на языке C#, который:

- называется Mutex и реализует двоичный семафор с помощью атомарной операции Interlocked.CompareExchange.

- обеспечивает блокировку и разблокировку двоичного семафора с помощью public-методов Lock и Unlock.

**public class Mutex**

**{**

**private Thread thread;**

**public void Lock()**

**{**

**Thread t = Thread.CurrentThread;**

**while(Interlocked.CompareExchange(ref thread, t, null) != null)**

**Thread.Yield();**

**Thread.MemoryBarrier();**

**}**

**public void Unlock()**

**{**

**Thread t = Thread.CurrentThread;**

**if (Interlocked.CompareExchange(ref thread, null, t) != t)**

**throw new SynchronizationLockException();**

**Thread.MemoryBarrier();**

**}**

**}**

Создать на языке C# класс ActionRunner, реализующий метод RunAndWaitAll, который:

- принимает в параметрах массив делегатов;

- выполняет все указанные делегаты параллельно с помощью пула потоков;

- дожидается окончания выполнения всех делегатов.

Привести простейший пример использования созданного класса.

**public class ActionRunner**

**{**

**int runningCount;**

**object sync = new object();**

**public void RunAndWaitAll(Action[] actions)**

**{**

**runningCount = actions.Length;**

**foreach (Action action in actions)**

**ThreadPool.QueueUserWorkItem(ExecuteAction, action);**

**lock (sync)**

**if (runningCount > 0)**

**Monitor.Wait(sync);**

**// Monitor.Exit(sync);**

**// Ожидание();**

**// Monitor.Enter(sync);**

**}**

**private void ExecuteAction(object state)**

**{**

**var action = (Action)state;**

**action();**

**lock (sync)**

**{**

**runningCount--;**

**if (runningCount == 0)**

**Monitor.Pulse(sync);**

**}**

**}**

**}**

Создать класс на языке C#, который:

- называется NativeBuffer и обеспечивает выделение и освобождение блока памяти ОС через функции Marshal.AllocHGlobal и Marshal.FreeHGlobal.

- содержит свойство Handle, позволяющее получить дескриптор блока памяти ОС;

- реализует метод Finalize для автоматического освобождения блока памяти;

- реализует интерфейс IDisposable для принудительного освобождения блока памяти.

**public class NativeBuffer : Object, IDisposable**

**{**

**private IntPtr handle;**

**private bool disposed;**

**public NativeBuffer(int size)**

**{**

**handle = Marshall.AllocHGlobal(size);**

**}**

**~NativeBuffer()**

**{**

**Dispose(false);**

**}**

**public IntPtr Handle**

**{**

**get**

**{**

**if (!disposed)**

**return handle;**

**else**

**throw new ObjectDisposedException(ToString());**

**}**

**}**

**public void Dispose()**

**{**

**if (!disposed)**

**{**

**Dispose(true);**

**GC.SuppressFinalize(this);**

**disposed = true;**

**}**

**}**

**protected virtual void Dispose(bool disposing)**

**{**

**//disposing = true - вызов уничтожения был сделан вручную, чистим //управляемые ресурсы сами**

**if (handle != IntPtr.Zero)**

**{**

**if (disposing)**

**{**

**//очистика управляемых ресурсов**

**}**

**Marshall.FreeHGLobal(handle);**

**}**

**}**

**}**

Реализовать консольную программу на языке C#, которая:

- принимает в параметре командной строки путь к сборке .NET (EXE- или DLL-файлу);

- загружает указанную сборку в память;

- выводит на экран полные имена всех public-типов данных этой сборки, упорядоченные по пространству имен (namespace) и по имени.

**using System.Reflection;  
  
if (args.Length == 1)  
{  
 try  
 {  
 var assembly = Assembly.LoadFrom(args[0]);  
 var types = assembly.GetTypes()  
 .Where(x => x.IsPublic)  
 .OrderBy(x => x.Namespace)  
 .ThenBy(x => x.FullName);  
 foreach (var type in types)  
 {  
 Console.WriteLine(type.FullName);  
 }  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e);  
 }  
}**