Вопросы для тестирования навыков работы с .Net 3.5 на языке C#3.0.

- Что будет выведено на экран в результате работы следующего кода(объясните результат):

Console.WriteLine("A" + "B" + "C");

Console.WriteLine('A' + 'B' + 'C');

Будет выведено:

ABC

198

В первом случае была произведена конкатенация строк, в результате которой была создана новая строка ABC, которая и была выведена на экран.

Во втором случае были просуммированы коды символов потому, что для типа char так определен оператор ‘+’.

- Какое значение присвоено x, если приведенный ниже код всегда выводит false:

float x = 0;

// ... some code

Console.Write(x == x);

Нужно присвоить константу float.NaN, это неопределенное значение и оно не проверяется сравнением. Для проверки есть метод IsNan

- Опишите отличия Value types от Reference types.

1) Хранят значения

2) Не могут быть null

3) Неявно наследуют System.ValueType

4) При присвоении одной переменной другой, создается копия значения, а не копируется ссылка.

- Раскритикуйте следующий код и предложите более эффективное по скорости решение.

if (o is Employee)

{

var e = (Employee)o;

}

Я бы написал так:

var e = o as Employee;

но тут по идее все равно нужна будет проверка на null, так что выигрыша в скорости я не вижу. Хотя запись короче и понятнее.

- Что будет выведено на экране в результате работы следующего кода:

public void DoWork()

{

int end = int.MaxValue;

int begin = end - 100;

int counter = 0;

for (int i = begin; i <= end; i++)

{

counter++;

}

Console.WriteLine(counter);

}

Ничего, так как после достижения счетчиком значения int.MaxValue произойдет еще одна инкрементация, которая добавит 1 в старший разряд и этим изменит знак на int.MinValue.

- Объясните результат работы следующего кода:

object obj = "Int32";

string str1 = "Int32";

string str2 = typeof(int).Name;

Console.WriteLine(obj == str1); // true

Console.WriteLine(str1 == str2); // true

Console.WriteLine(obj == str2); // false !?

Компилятор прооптимизирует этот код и создаст строковую константу “Int32”, которая будет присвоена obj и str1. Поэтому они имеют одинаковые ссылки и при сравнении, как object, вернут true. Однако str2 получена другим способом и указывает на другой экземпляр строки. Строки str1 и str2 будут сравниваться по значению и, следовательно, вернут true.

- Будет ли выполнен код в блоке finally в следующем случае?

try

{

Console.WriteLine("In Try block");

return;

}

finally

{

Console.WriteLine("In Finally block");

}

Да, блок finally выполняется всегда.

- Почему явно реализуемый программистом статический конструктор типа может снизить скорость работы приложения?

Явно реализуемый статический конструктор не помечается флагом BeforeFieldInit, в результате конструктор будет выполняться непосредственно перед первым обращением к классу, а проверки на это обращение займут дополнительное время. В реальности до четвертого фраймворка статические конструкторы с флагом BeforeFieldInit выполняются перед первым методом приложения, так что, добавляя явный статический конструктор, можно обеспечить отложенную инициализацию.

- Почему следующий код не может скомпилироваться?

var str = "Hello world";

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

{

str[i] = str[str.Length - 1 - i];

}

Потому что индексатор строки доступен только для чтения.

- Перечислите основные особенности алгоритма работы сборщика мусора CLR .Net, которые влияют на скорость работы приложения.

1) Поколения объектов

2) Дефрагментация кучи

3) Большие объекты лежат в отдельной куче и не дефрагментируются

4) Приостановка потоков на время удаления объектов из кучи

5) Вызов деструктора отдельным потоком

- В вашем .Net-приложении объявлена константа 0x0A0B0C0D (шестнадцатеричная система) типа Int32. Допустим, она записана в четыре последовательно расположенные ячейки в памяти компьютера. Каково содержимое каждой из этих ячеек?

00001010 00001011 00001100 00001101

Старший бит знаковый - 0 плюс, 1 минус.

- Раскритикуйте следующий фрагмент кода.

public class MoneyMover

{

//...

public void Transfer(Account sourceAccount, Account targetAccount, Money amountOfMoney)

{

try

{

\_db.Transfer(sourceAccount, targetAccount, amountOfMoney);

}

catch

{

}

}

}

Невозможно узнать было ли выброшено исключение и что это было за исключение, не говоря о том, что если это экономия на проверках, то операция try catch относительно медленная и эффективней проверить счета, например, на null до вызова метода Transfer.

- Какие нововведения .NET 3.0 и C# 3.0 позволили использовать следующую запись?

var x = from p in Prices

where p.Value > 10

select new { Price = p.Value };

1. Linq
2. Анонимные методы
3. Анонимные типы
4. Методы расширения
5. Выведение типа (var)

- Реализуйте метод, который вернет минимальный из максимальных элементов входных последовательностей.

public int MinMax(IEnumerable<IEnumerable<int>> enumerables)

{

return enumerables.Select(e=> e.Max()).Min();

}

- Класс Record представляет собой запись о цене продукта в каждый момент времени. Реализуйте метод, который из набора записей выбирает элементы с наибольшим значением цены за каждый день (см. сигнатуру метода ниже). Предполагается, что цена меняется в течение дня.

class Record

{

public DateTime DateTime { get; set; }

public decimal Price { get; set; }

}

public static IEnumerable<Record> RecordsWithMaxPricePerDay(IEnumerable<Record> records)

{

return records.GroupBy(r => r.DateTime.Date)

.Select(g => g.Aggregate((max, next) => max.Price < next.Price ? next : max));

}

- Что будет выведено на экран в результате работы следующего кода(объясните результат):

var i = 0;

var list = new[] { 0, 1, 2 };

var a = list.Select(x => (x + i++).ToString());

Console.WriteLine(string.Join(" ", a.ToArray()));

Console.WriteLine(string.Join(" ", a.ToArray()));

0 2 4

3 5 7

Метод селект возвращает выражение, которое будет выполнено при каждом переборе, в данном случае при вызове метода ToArray().

- Раскритикуйте следующий код:

Thread thread = new Thread(...);

thread.Start();

// ...

thread.Abort();

Не рекомендуется использовать метод Abort, так как он вызывает асинхронное исключение ThreadAbortException, которое может произойти в любой точке кода, например, в статическом конструкторе.

- Предположим, что перед вами кусок кода многопоточного приложения. Раскритикуйте этот код:

public class FooBar

{

public event EventHandler SomethingHappened;

public void DoWork()

{

// ...

if (SomethingHappened != null)

{

SomethingHappened(this, EventArgs.Empty);

}

}

}

Между проверкой на наличие подписчиков и вызовом события, подписчик может отписаться. Можно воспользоваться атомарной операцией так:

var somethingHappened = SomethingHappened;

if (somethingHappened != null)

somethingHappened (this, EventArgs.Empty);

- Чему эквивалентна следующая конструкция?

lock(\_lockObject)

{

// ...

}

До четвертого фреймворка:

var temp = syncRoot;

Monitor.Enter(temp);

try

{

//…

}

finally

{

Monitor.Exit(temp);

}

В четвертом фраймворке:

bool lockWasTaken = false;

var temp = syncRoot;

try

{

Monitor.Enter(temp, ref lockWasTaken);

{

// …

}

}

finally

{

if (lockWasTaken)

Monitor.Exit(temp);

}

- Для чего используется ключевое слово volatile?

Указывает, что поле может быть изменено несколькими потоками. Эти поля не проходят оптимизацию компилятором, что гарантирует наличие в них наиболее актуального значения. Используется в полях, к которым обращаются из разных потоков без блокировки.

- Три потока встали в очередь на получение блокировки, вызвав метод Monitor.Enter(criticalSection). Сначала встал в очередь поток A, затем -- поток B, следом – поток C. В каком порядке потоки получат блокировку?

В порядке получения блокировки: ABC, то есть FIFO

- Раскритикуйте следующий код:

lock(this) {...}

Внешний код также может использовать экземпляр этого класса в качестве объекта синхронизации, что вызовет DeadLock при обращении к этому участку кода. В качестве объектов синхронизации рекомендуется использовать private поля.

- Что и почему будет записано в локальную переменную x в результате работы следующего кода:

static void DoWork()

{

var threads = Enumerable.Range(0, 50).Select(i => new Thread(ThreadFunc)).ToList();

foreach (var t in threads)

t.Start();

foreach (var t in threads)

t.Join();

Console.WriteLine(x);

}

static volatile int x;

static void ThreadFunc()

{

for (int i = 0; i < 10000; i++)

x++;

}

Будет выведено 500 000 потому, что переменная x помечена, как volatile, следовательно, в ней всегда содержатся последние значения и сбоев в инкременте не будет.

- Что такое транзакция?

Набор действий, которые должны выполниться все вместе либо не выполниться совсем. В классическом представлении транзакция должна удовлетворять принципам ACID (атомарность, согласованность, изолированность, долговечность).