**Михайловский Александр Станиславович:**

1. Что такое управляемый/неуправляемый код?  
     
   Управляемый код компилируется в промежуточный байт-код(CLR), неуправляемый код напрямую выполняется на операционной системе. В неуправляемом коде мы сами отвечаем за управление памятью и ресурсами.
2. Опишите устройство памяти в **С#.**Память управляется автоматически с помощью сборщика мусора. Основные компоненты памяти:  
   Куча - это область памяти, где выделяются объекты и данные во время выполнения программы. Это управляемая и динамически расширяемая область памяти. Предназначена для хранения объектов, которые создаются во время выполнения программы   
   Стек - используется для хранения локальных переменных и данных.  
   Сборщик мусора - отслеживает и управляет объектами, выделенными на куче. Он автоматически освобождает память, занятую объектами, которые больше недоступны/не используются
3. Что такое **CLR**?  
     
   CLR - это среда выполнения, которая обеспечивает исполнение и управление управляемым кодом.
4. В чем отличие ссылочного типа от значимого?  
     
   Переменные ссылочного типа хранят ссылку на объект, в то время как значимый тип хранит сами значения. При присваивании переменной ссылочного типа другой переменной, копируется ссылка, а не сам объект. В случае передачи значимого типа - копируется значение. Объекты ссылочного типа создаются на куче, а ссылка на них хранится в переменной. Значимые типы хранятся на стеке.  
   default:  
   ссылочные типы - null  
   значимый - null невозможно, так как они хранят какое-то значение(в зависимости от типа данных).

1. Что такое явное/неявное преобразование типов?

Неявное преобразование происходит автоматически, когда значение одного типа данных присваивается переменной другого типа, который может содержать более широкий диапазон значений.

Неявное преобразование:  
 int n1 = 1;

double n2 = n1;

Явное преобразование производится вручную с использованием оператора приведения типов. В данном случае увеличивается риск ошибок, а также потери данных

Явное преобразование:

double n1 = 3.14;

int n2 = (int)n1;

1. Что такое операция упаковки/распаковки?  
   Операция упаковки и распаковки связаны с преобразованием переменных ссылочного типа в значимые и обратно

boxing:

int n1 = 1;

object object1 = n1;

После упаковки переменная object1 имеет ссылку на объект.

unboxing:

object object1 = 1;

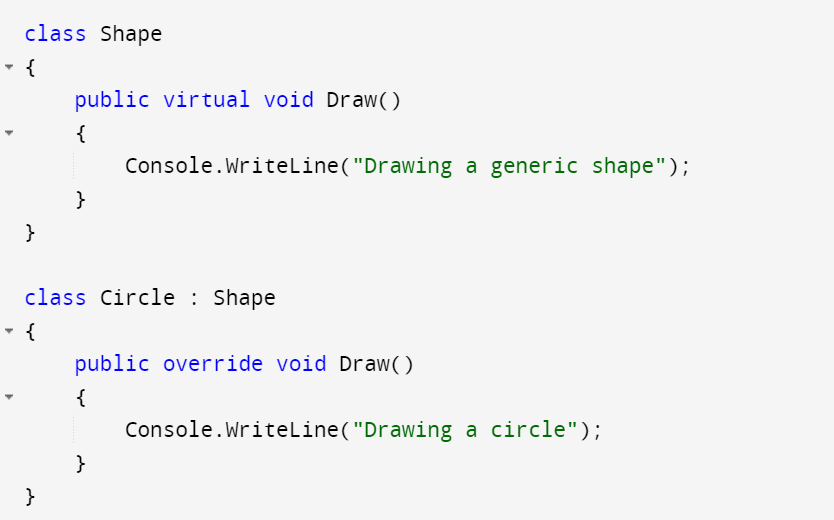
int n1 = (int)object1;

После распаковки переменная n1 имеет значение.

1. В чём отличие метода **Finalize** от **Dispose**?   
   Метод **Finalize** является частью механизма сборки мусора. Он вызывается автоматически сборщиком мусора перед оканчательным удалением объекта из памяти, но точное время вызова не определено и зависит от сборщика мусора.

Метод Dispose предназначен для явного освобождения ресурсов, как управляемых так и неуправляемых. Этот метод не является частью сборщика мусора, и требует явного вызова кодом. Dispose предоставляет бОльшую гибкость и контроль над процессом освобождения мусора по сравнению с Finalize.

1. Если сделать **return** из **catch** выполнится ли **finally**?  
     
   Блок finally выполняется всегда, независимо от того, произошло исключение или нет, и независимо от наличия оператора return
2. Как можно передавать параметры методу в **С#**? Модификаторы параметров.  
     
   Параметры методу можно передавать по значению, ссылке или с использованием модификатором параметров.
3. Что такое **dynamic**? Как оно работает?  
     
   Это ключевое слово, которое позволяет создавать переменные и объекты, тип которых определяется во время выполнения программы, а не на моменте компиляции.
4. Какие принципы ООП использованы в коде ниже

  
Наследование, инкапсуляция.

1. Назовите отличия между интерфейсом и абстрактным классом?  
     
   Абстрактные классы могут содержать конструкторы, в то время как интерфейс не может. Абстрактный класс может хранить данные в полях, интерфейс не может хранить данных. При использовании абстрактных классов не может быть множественного наследования, При использовании интерфейсов класс может наследоваться от нескольких интерфейсов. Абстрактный класс может переопределять только некоторые элементы, а интерфейс должен реализовывать все элементы. Абстрактный класс - наследование только для классов, интерфейс может быть реализован структурой.

1. В чем отличие **delegate**, **Action**, **Func** и **Predicate**?  
   delegate - это тип, который предоставляет ссылку на метод с определённой сигнатурой(возвращаемый тип и параметры).Может представлять метод с любым количеством параметров и любым типом возвращаемого значения.  
   action - это предопределённый делегат, который предоставляет метод без возвращаемого значения.Делегат Func представляет собой метод, который возвращает значение. Он используется, когда требуется передать метод, возвращающий значение, например для вычислений.Делегат Predicate представляет собой метод, который принимает значение и возвращает булевое значение. Он используется для проверки условий.
2. Для чего в **.NET** используется конструкция **using(…){…}**, и во что она разворачивается?   
     
   Эта конструкция используется для работы с объектами, которые реализуют интерфейс IDisposable. Обеспечивает автоматическое и надёжное освобождение ресурсов. Разворачивается в блок try-finally
3. Синхронизация потоков, какие примитивы синхронизации вы знаете (не менее 4).  
     
   Монитор, семафор, мьютекс, autoresetevent.
4. Коротко опишите алгоритм работы сборщика мусора?  
   Сборщик мусора обходит все корневые объекты и каждый достижимый объект помечается как “живой” - это говорит о том, что объект всё ещё используется.   
   После маркировки сборщик мусора проходится по всей куче и освобождает память, не помеченными объектами. Далее он перемещает все живые объекты, чтобы они занимали непрерывные участки памяти.
5. При работе с неуправляемым кодом, какие механизмы используются для работы с памятью (методы, классы, конструкции, паттерны)?  
     
   -
6. Отличаются ли **IEnumerable<T>** и **IQueryable<T>**? Если да, в чем отличие?  
    IEnumerable<T> и IQueryable<T> представляют разные интерфейсы в .NET, используемые для работы с коллекциями данных.

Выполнение операций:

IEnumerable<T> представляет интерфейс для коллекций, которые могут быть перечислены (итерированы) последовательно.

Операции над IEnumerable<T> выполняются в памяти на клиентской стороне.

Операции над IEnumerable<T> выполняются на основе LINQ to Objects и обычно подходят для работы с коллекциями, хранящимися в памяти.

Запросы к источникам данных:

IQueryable<T> расширяет IEnumerable<T> и предоставляет интерфейс для создания и выполнения запросов к источникам данных (например, базам данных).

Операции над IQueryable<T> переводятся в язык запросов соответствующего провайдера (например, LINQ to SQL, LINQ to Entities) и выполняются на стороне сервера.

Это позволяет использовать оптимизации, предоставляемые провайдером запросов, такие как фильтрация и сортировка на уровне базы данных.

Отложенное выполнение запросов:

IEnumerable<T> выполняет операции немедленно при вызове методов запросов, таких как Where, Select, OrderBy и т.д.

IQueryable<T> поддерживает отложенное выполнение запросов. Запросы не выполняются немедленно, а формируют дерево выражений, которое будет выполнено при доступе к результатам.

Возможности оптимизации запросов:

IQueryable<T> позволяет провайдерам запросов оптимизировать выполнение запросов, выполняя фильтрацию, проекцию и другие операции на стороне источника данных.

IEnumerable<T> не предоставляет таких возможностей оптимизации на уровне источника данных.

Основное отличие между IEnumerable<T> и IQueryable<T> заключается в том, что IQueryable<T> предоставляет более мощные возможности для создания и выполнения запросов к источникам данных, в то время как IEnumerable<T> просто позволяет перечислять коллекции в памяти.

1. Можно ли реализовать в одном классе два интерфейса описывающих методы с одинаковой сигнатурой.  
     
   Да, в одном классе можно реализовать два или более интерфейса, даже если они содержат методы с одинаковой сигнатурой. Однако, следует учитывать, что при наличии методов с одинаковой сигнатурой в разных интерфейсах, класс должен явно указывать, какой интерфейс он имплементирует при реализации метода.
2. Отличаются ли Асинхронность и Многопоточность? Если да, в чем отличие?  
     
   Да, асинхронность и многопоточность представляют различные концепции в программировании.

Цель:

- Многопоточность предназначена для повышения производительности и использования ресурсов компьютера путем параллельного выполнения задач.

- Асинхронность призвана улучшить отзывчивость и отзывчивость программы, особенно при выполнении операций ввода-вывода или операций сети, которые могут быть блокирующими.

Модель программирования:

- Многопоточность обычно требует явного создания и управления потоками, синхронизации доступа к общим ресурсам и управления состоянием потоков.

- Асинхронность использует асинхронные операции, колбэки, задачи или другие механизмы, которые позволяют коду продолжать выполнение без ожидания завершения операции.

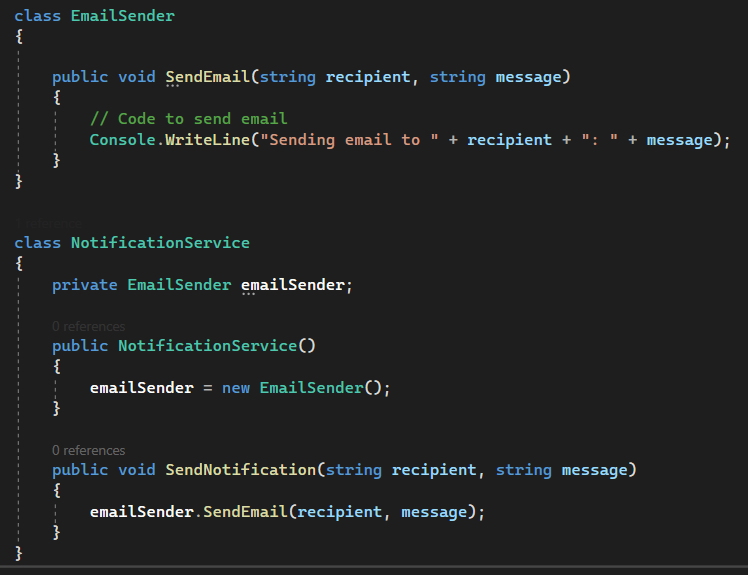
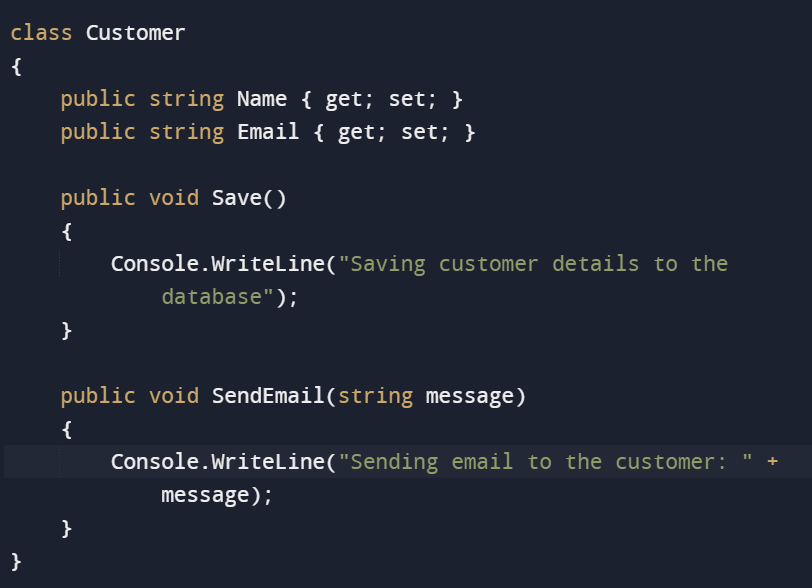
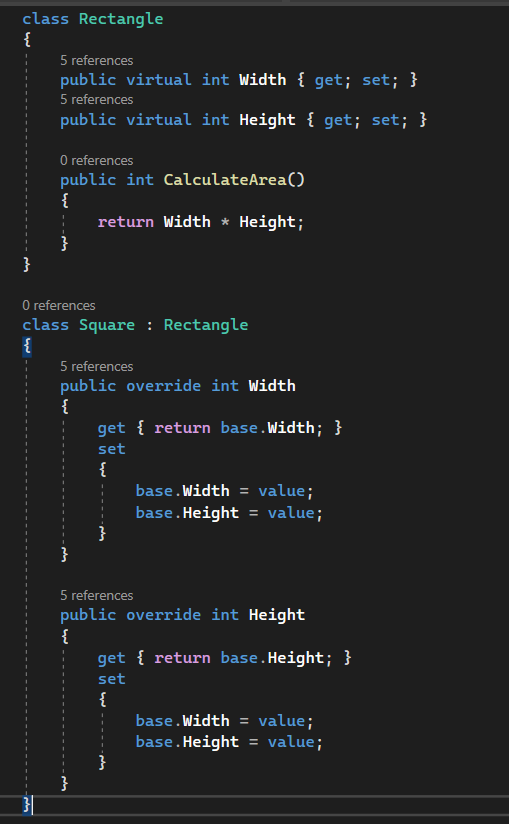
Параллелизм:

- Многопоточность может предоставлять параллелизм, то есть выполнение задач одновременно на нескольких процессорных ядрах.

- Асинхронность не обязательно предоставляет параллелизм, но может выполнять операции ввода-вывода или задачи параллельно, освобождая поток для выполнения других задач.

Основное отличие между асинхронностью и многопоточностью заключается в их целях и подходах. Многопоточность связана с одновременным выполнением задач на нескольких потоках, в то время как асинхронность относится к выполнению операций без блокировки и ожидания результатов, чтобы улучшить отзывчивость программы.

1. Какие принцип SOLID нарушены в коде? (1 принцип в каждом примере кода)



–

Задача 1:

Разработайте **метод расширения** для типа **int**, возвращающий значение в виде строки.

Пример: Возвращает «1234» в виде «one two three four»

Задача 2:

Объявить коллекцию элементов типа **int**. Инициализировать его вручную или random. При помощи **LINQ** отсортировать коллекцию по возрастанию и вернуть вторую половину коллекции (округляя вверх если число элементов нечетное) отсортированную по убыванию, где каждый элемент будет возведен в квадрат.

Задача 3:

Разработайте класс содержащий произвольный публичный метод и событие. После 10 вызовов метода у любых экземпляров разработанного класса необходимо вызвать событие и информировать об этом пользователя. Обработку события произвести либо при помощи анонимного метода, либо при помощи лямбда-выражения.

Задача 4:

Разработайте класс для выполнения математических функций (факториал, возведение в степень) не использую Math. Необходимо предусмотреть защиту от некорректного ввода данных, а также обработку исключительных ситуаций. При возникновении исключительной ситуации пользователь должен получить соответствующее сообщение с описанием исключительной ситуации (custom error). Не забывайте про принципы OOP / SOLID.

Задача 5:

Предположим, у нас есть список учащихся с их именами и возрастом, и мы хотим отфильтровать учащихся моложе определенного возраста, сгруппировать их по возрастному диапазону и, наконец, получить количество учащихся в каждой возрастной группе.

*class Student*

*{*

*public string Name { get; set; }*

*public int Age { get; set; }*

*}*

*List<Student> students = new List<Student>()*

*{*

*new Student { Name = "John", Age = 20 },*

*new Student { Name = "Alice", Age = 22 },*

*new Student { Name = "David", Age = 18 },*

*new Student { Name = "Sarah", Age = 21 },*

*new Student { Name = "Michael", Age = 19 },*

*new Student { Name = "Emily", Age = 20 },*

*};*

*int minAge = 20;*

*Вывод:*

*Age Range: 20 - 24*

*Count: 4*

Задача 6:

Реализуйте класс Logger, который может логировать сообщения различного типа реализующие интерфейс ILoggable и использующий определенную стратегию логирования (всего 2 стратегии ConsoleLoggingStrategy, FileLoggingStrategy у обоих классов один публичный метод LogMessage). Стратегия логирования отвечает за канал вывода такие как консоль или файл, основываясь на выбранной стратегии.

interface ILoggable

{

public string ToString();

}

Структура класса Logger

public void Log(T message)

private string FormatMessage(T message)