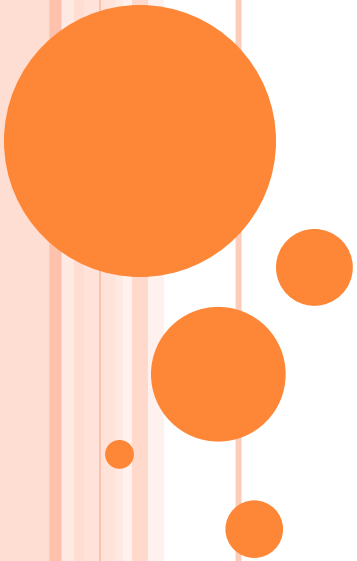
Самостоятельная работа. ответ

1. Структуры относятся к **значимым типам** (value type).
2. Это значит, что они неявно унаследованы от класса **System.ValueType**.
3. К типам-значениям относятся базовые (примитивные) типы данных такие как **short, int, uint, double, float, bool и другие.**
4. Если объявляется переменная (экземпляр) структурного типа, то этот экземпляр размещается в памяти, которая называется **стек.**
5. При объявлении структуры используется ключевое слово **struct**.
6. По умолчанию, всем членам структуры устанавливается модификатор доступа **private**. Это означает, что при объявлении переменной типа Структура, невозможно будет обратиться к полям структуры непосредственно.
7. Для доступа к полям структуры, используются следующие модификаторы **public**.
8. Структуры не поддерживают **наследование**, поэтому их члены нельзя указывать как abstract, virtual, protected.
9. Объект, структуры может быть создан с помощью оператора **new** таким же образом, как и объект класса, но в этом нет необходимости.
10. Оператор **new** вызывает конструктор, используемый по умолчанию.
11. Конструктор по умолчанию это конструктор,. который не имеет никаких **параметров**
12. В структурах допускается определять конструктор, но не **деструкторы**
13. Структуры используются, когда нужно **сгруппировать** небольшие объемы данных.
14. Структуры сохраняются в **стеке**, поэтому выделение/освобождение памяти происходит быстрее.
15. Поскольку структуры принадлежат **к типам-значениям**, то при использовании структур используется меньший объем памяти. В случае с классами нужно использовать дополнительную ссылочную переменную.



Пользовательские типы данных

Отладка и обработка исключительных ситуаций

Конструкция try..catch..finally

Что такое исключение?

Исключения (Exceptions) это тип ошибки, которая происходит при выполнении приложения. Ошибки обычно означают появление неожиданных проблем.

Исключения - это ожидаемые ошибки, обработка которых организована в коде.

Типы исключений. Класс Exception

- Базовым для всех типов исключений является тип **Exception**. Этот тип определяет ряд свойств, с помощью которых можно получить информацию об исключении:
- **InnerException**: хранит информацию об исключении, которое послужило причиной текущего исключения
- **Message**: хранит сообщение об исключении, текст ошибки
- **Source**: хранит имя объекта или сборки, которое вызвало исключение
- **StackTrace**: возвращает строковое представление стека вызывов, которые привели к возникновению исключения *(**трассировка стека** – **это** список методов, которые были вызваны до момента, когда в приложении произошло исключение)*
- **TargetSite**: возвращает метод, в котором и было вызвано исключение

Специализированные типы исключений, которые предназначены для обработки каких-то определенных видов исключений

- **DivideByZeroException**: представляет исключение, которое генерируется при делении на ноль
- **ArgumentOutOfRangeException**: генерируется, если значение аргумента находится вне диапазона допустимых значений
- **ArgumentException**: генерируется, если в метод для параметра передается некорректное значение
- **IndexOutOfRangeException**: генерируется, если индекс элемента массива или коллекции находится вне диапазона допустимых значений
- **InvalidCastException**: генерируется при попытке произвести недопустимые преобразования типов
- **NullReferenceException**: генерируется при попытке обращения к объекту, который равен null (то есть неопределен)

Обработка исключений. конструкция try…catch…finally

```
try (пытаться)
{

}
catch (ловить)
{

}
finally (наконец-то)
{

}
```

При использовании блока **try...catch..finally** вначале выполняются все инструкции в блоке **try**. Если в этом блоке не возникло исключений, то после его выполнения начинает выполняться блок **finally**. И затем конструкция try..catch..finally завершает свою работу.

Если же в блоке try вдруг возникает исключение, то обычный порядок выполнения останавливается, и среда CLR начинает искать блок **catch**, который может обработать данное исключение. Если нужный блок catch найден, то он выполняется, и после его завершения выполняется блок finally.

Если нужный блок catch не найден, то при возникновении исключения программа аварийно завершает свое выполнение.

Common Language Runtime (англ. **CLR** — общеязыковая исполняющая **среда**) — исполняющая **среда** для байт-кода CIL (MSIL), в который компилируются программы, написанные на . NET-совместимых языках программирования (C#, Managed C++, Visual Basic . NET, F# и др).

Пример:

В данном случае происходит деление числа на 0, что приведет к генерации исключения типа **System.DivideByZeroException**.

```
class Program
{
    static void Main(string[] args)
    {
        int x = 5;
        int y = x / 0;
        Console.WriteLine($"Результат: {y}");
        Console.WriteLine("Конец программы");
        Console.Read();
    }
}
```

Чтобы избежать подобного аварийного завершения программы, следует использовать для обработки исключений конструкцию **try...catch...finally**.

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            int x = 5;
            int y = x / 0;
            Console.WriteLine($"Результат: {y}");
        }
        catch
        {
            Console.WriteLine("Возникло исключение!");
        }
        finally
        {
            Console.WriteLine("Блок finally");
        }
        Console.WriteLine("Конец программы");
        Console.Read();
    }
}
```

Опять же возникнет исключение в блоке try, так как мы пытаемся разделить на ноль. И дойдя до строки int y = x / 0; выполнение программы остановится. CLR найдет блок **catch** и передаст управление этому блоку.
После блока catch будет выполняться блок finally.

Результат:
Возникло исключение!" Блок finally
Конец программы

Генерация исключения и оператор throw

Обычно система сама генерирует исключения при определенных ситуациях, например, при делении числа на ноль.

Но язык C# также позволяет генерировать исключения вручную с помощью оператора **throw**.

С помощью этого оператора можно создать исключение и вызвать его в процессе выполнения.

Пример: в программе происходит ввод строки. если длина строки будет больше 6 символов, должно возникать исключение:

```
static void Main(string[] args)
{
    try
    {
        Console.Write("Введите строку: ");
        string message = Console.ReadLine();
        if (message.Length > 6)
        {
            throw new Exception("Длина строки больше 6 символов");
        }
    }
    catch (Exception e)
    {
        Console.WriteLine($"Ошибка: {e.Message}");
    }
    Console.Read();
}
```

*После оператора **throw** указывается объект исключения, через конструктор которого передается сообщение об ошибке.*

В блоке catch сгенерированное исключение будет обработано.

Свойства исключений

- Исключения представляют собой типы, производные от System.Exception.
- Блок try используется для выполнения таких инструкций, которые могут создавать исключения.
- Когда внутри такого блока try возникает исключение, поток управления переходит к первому подходящему обработчику исключений в стеке вызовов. В C# ключевое слово catch (ловить) обозначает обработчик исключений.

МАТЕРИАЛЫ

- https://metanit.com/sharp/tutorial/2.14.php
  Обработка исключений