



C# Professional

Исключения и
нюансы работы с ними



Проверить, идет ли запись

Меня хорошо видно && слышно?



Ставим "+", если все хорошо
"-", если есть проблемы



Тема вебинара

Исключения и нюансы работы с ними

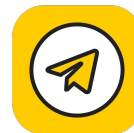


Михаил Дмитриев

Ведущий программист НИПК Электрон

Разрабатываю и поддерживаю приложения для работы с радиологическими комплексами

<https://t.me/sf321>



Правила вебинара



Активно
участвуем



Off-topic обсуждаем
в учебной группе

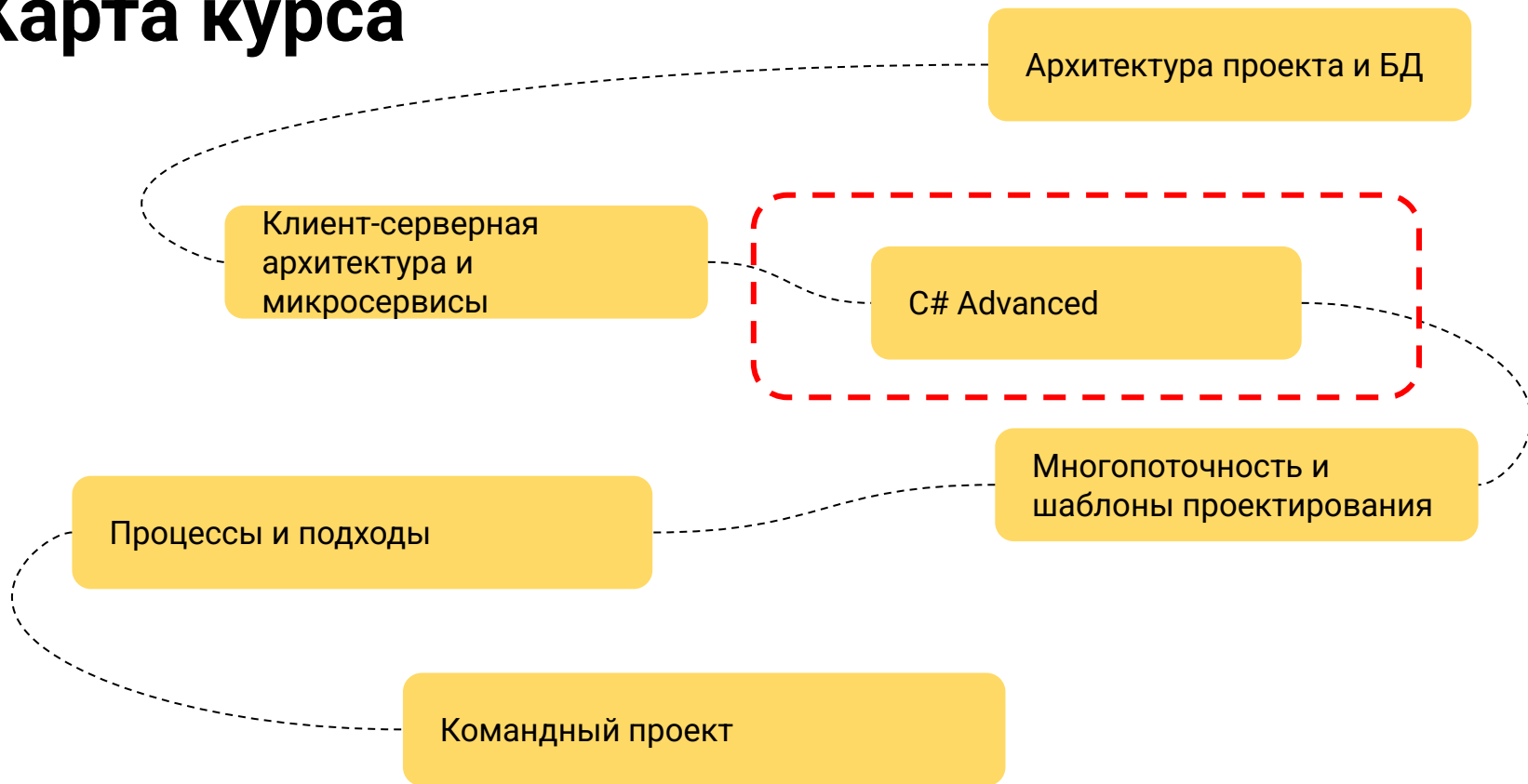


Задаем вопрос
в чат или голосом



Вопросы вижу в чате,
могу ответить не сразу

Карта курса



Маршрут вебинара

Знакомство

Что такое исключение и как
использовать

throw, try, catch, finally

Порядок перехвата

Условные исключения

Примеры

Рефлексия

Цели вебинара

К концу занятия вы сможете

1. Изучить понятие исключение и поработать с примерами
2. Использовать средства языка для работы с исключениями
3. Обработать нестандартные случаи ошибок
4. Создавать свои собственные исключения и обрабатывать их



Смысл

Зачем вам это уметь

1. Уметь корректно обрабатывать ошибки в работе ваших программ
2. Уметь определять и работать не только со стандартными исключениями
3. Уметь быстро находить источник возникновения ошибки в вашей программе
4. Использовать Best Practices при работе с исключениями в ваших программах



Тестирование



Исключения (Exception)



Что такое исключение?

Определения, общая информация

Исключение (exception) – событие, означающее, что в программе что-то пошло не так (возникла ошибка)

Выбросить (пробросить) исключение – вызвать событие исключения

Некоторые ситуации возникновения исключений:

- Поделили на ноль
- Аргумент функции некорректный
- Доступ к полю неинициализированного объекта
- Выход за границы массива

Где могут появиться:

- Во время выполнения .NET Runtime (доступ к пустому объекту)
- Выбрасываются сторонними библиотеками
- Вызываются пользователем принудительно

Как выглядят?

```
var a = 0;  
var b = 4;  
Console.WriteLine(b / a);
```

```
Unhandled exception. System.DivideByZeroException: Attempted to divide by zero.  
   at Otus.Exceptions.Program.Main(String[] args) in C:\Users\Эдгар\Documents\otus\modules\csharp-base\8 - Исключения  
и их обработка\code\Otus.Exceptions\Otus.Exceptions\Program.cs:line 11
```

LIVE

Исключения и управление состоянием

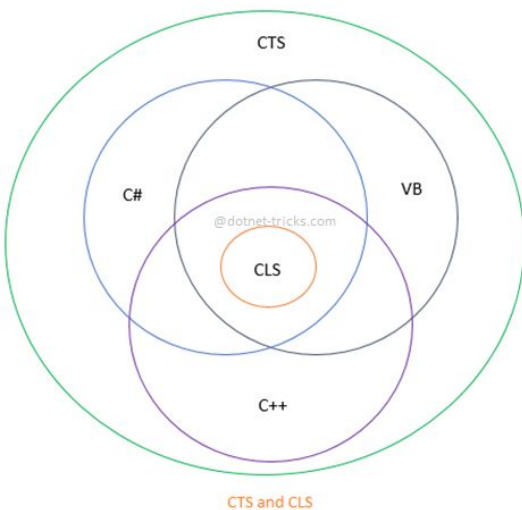
Конструируя тип (Классы) мы заранее представляем ситуации в которых он будет использоваться. В качестве имен существительные. Затем определяются свойства события, методы и тп . Форма определения этих членов становится интерфейсом типа. Именно члены определяют допустимые действия с типом и его экземплярами. Если член сборки не может решить возложенную на него задачу программа выбрасывает исключение.

```
C#  
internal sealed class Account{  
    public static void Transfer(Account from, Account to, Decimal amount){  
        from-=amount;  
        to+=amount;  
    }  
}
```

Механизм обработки исключений

```
C#
private void SomeMethod(){
    try{
        // Код, требующий корректно восстановления
        // или очистки ресурсов
    }
    catch(InvalidOperationException){
        // Код восстановления работоспособности
        // после InvalidOperationException
    }
    catch(IOException){
        // Код восстановления работоспособности
        // после IOException
    }
    catch{
        // Код восстановления работоспособности после всех остальных исключений
        // После перехвата исключений их обычно генерируют повторно
        // Рассмотрим эту тему позже
        throw;
    }
    finally{
        //Здесь находится код, выполняющий очистку ресурсов
        //после операций начатых в блоке try. Этот код
        //выполняется ВСЕГДА вне зависимости от наличия исключения
    }
    //Код, следующий за блоком finally, выполняется, если в блоке try
    //не генерировалось исключение или если исключение было перехвачено
    //блоком catch, а новое не генерировалось
}
```


CLS-совместимые и CLS-несовместимые исключения



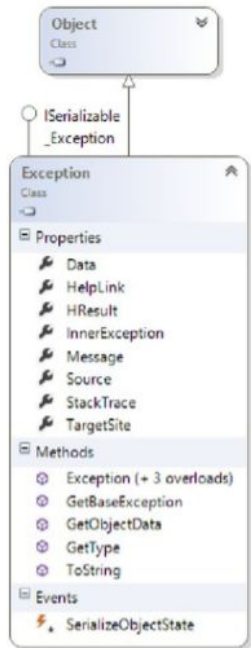
```
private void SomeMethod(){  
    try{  
        // Код, требующий корректно восстановления  
        // или очистки ресурсов  
    }  
    catch(Exception e){  
        // До C# 2.0 этот блок перехватывал только CLS-совместимые исключения  
        // В C# это блок научился перехватывать также  
        // CLS- несовместимые исключения  
    }  
    catch{  
        // Во всех версиях C# данный блок перехватывает  
        // и совместимые, и не совместимые с CLS исключения  
        throw; // повторная генерация перехваченного исключения  
    }  
}
```

C#

Как работать с исключением

System.Exception

System.Exception



Exception is a base class for all exceptions

Important properties:

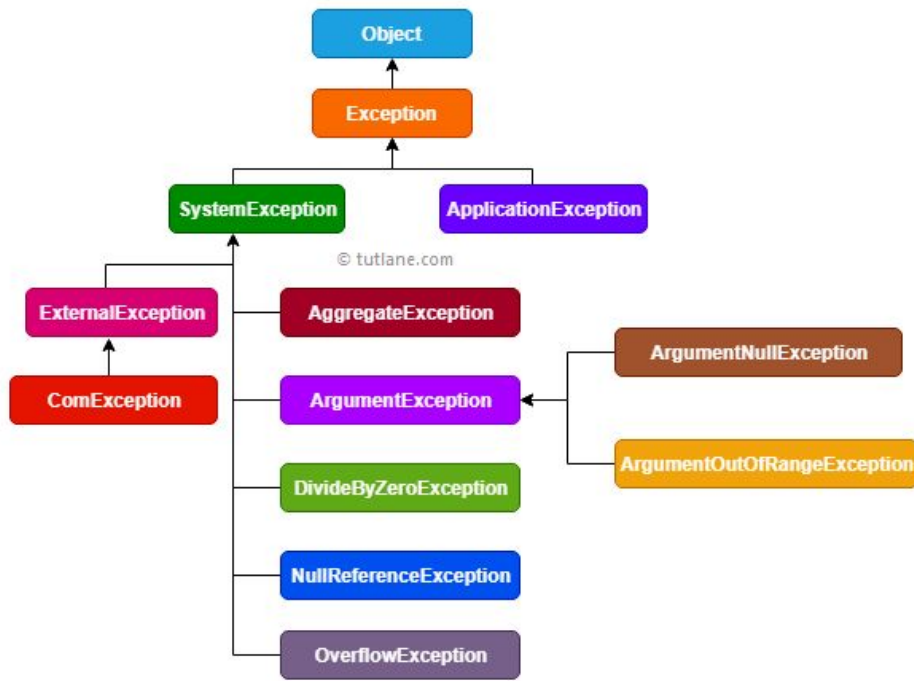
- **Message** – user-oriented message about error
- **Source** – name of an error source (application or object)
- **InnerException** – inner exception (if called from other)
- **StackTrace** – call stack to the point of exception call
- **TargetSite** – method name which raised an exception
- **HelpLink** – URL-address to information about exception
- **Data** – dictionary with additional information with exception (IDictionary)

System.Exception – базовый класс исключений
Все исключения наследуются на определенном уровне от System.Exception

Принято все классы исключений называть с суффиксом Exception

- InvalidOperationException**Exception**
- OutOfRange**Exception**
- ArgumentNullException

Иерархия исключений



Генерация собственных исключений

```
/// <summary>  
/// Мое собственное исключение  
/// </summary>  
class MyCystomException : Exception  
{  
  
}
```

LIVE

Варианты перехвата

- try catch
- try catch finally
- `AppDomain.CurrentDomain.UnhandledException`
- `Application.ThreadException` (WinForms, WPF)

Stacktrace

- Содержит цепочку вызовов методов до места, где произошло исключение (с точностью номера строки)
- Можно получить и `Environment.StackTrace`
- Можно использовать `System.Diagnostics.StackTrace`

LIVE

Операторы

Общее, синтаксис

```
try
{
    if (b == 0)
        throw new DivideByZeroException("Делим на ноль");
    return a / b;
}
catch (Exception e)
{
    Console.WriteLine("Произошла ошибка");
    return 0.0;
}
finally
{
    Console.WriteLine("Я блок finally ");
}
```

throw – оператор вызова исключения

```
try
{
    if (b == 0)
        throw new DivideByZeroException("Делим на ноль");
    return a / b;
}
catch (Exception e)
{
    Console.WriteLine("Произошла ошибка");
    return 0.0;
}
finally
{
    Console.WriteLine("Я блок finally ");
}
```

Синтаксис

throw *объект_класс_exception_или_производного*

throw , варианты использования

Можно так

```
throw new Exception("Я сообщение об ошибке");
```

А можно так

```
var ex = new Exception("Я сообщение об ошибке");  
ex.Data.Add("a", "2");  
throw ex;
```

try catch

```
try
{
    if (b == 0)
        throw new DivideByZeroException("Делим на ноль");
    return a / b;
}
catch (Exception e)
{
    Console.WriteLine("Произошла ошибка");
    return 0.0;
}
finally
{
    Console.WriteLine("Я блок finally ");
}
```

try catch

try catch – операторы перехвата исключений

try {} – объявляет область кода, где потенциально может выбрасываться исключение

catch{} – «ловит» исключение из блока try и обрабатывает данную ситуацию

Синтаксис:

```
try
{
    // Здесь может быть какой-то код
}
catch (Exception exc)
{

    // А здесь происходит обработка исключения exc
    // вызванного в каком-то коде
}
```


try catch ловим разные типы исключений

```
try
{
    // Здесь может быть какой-то код
}
catch (FooException exc)
{
    // Если какой-то код выкинул FooException
}
catch (BarException exc)
{
    // Если какой-то код выкинул BarException
}
```

try catch кидаем дальше

```
try
{
    // Здесь может быть какой-то код
}
catch (Exception exc)
{
    ОбработкаИсключения(exc);

    // пробрасываем то же исключение
    throw;
}
```

LIVE

finally

```
try
{
    if (b == 0)
        throw new DivideByZeroException("Делим на ноль");
    return a / b;
}
catch (Exception e)
{
    Console.WriteLine("Произошла ошибка");
    return 0.0;
}
finally
{
    Console.WriteLine("Я блок finally ");
}
```

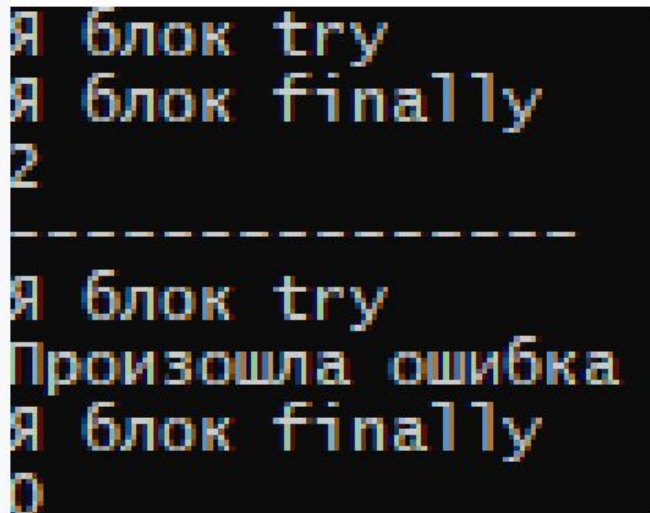
finally

- **finally** – оператор выполнения кода после всех обработок
- может использоваться конструкция **try{}finally{} без catch{}**
- Блок внутри **выполняется всегда**, есть исключение или нет
- в блоке **нельзя** возвращать что-то (**return**)
- используется в различных инструкциях представляющих синтаксический сахар lock, using,foreach, при определении деструктора

finally

```
double Divide(int a, int b)
{
    try
    {
        Console.WriteLine("Я блок try");
        return a / b;
    }
    catch (Exception)
    {
        Console.WriteLine("Произошла ошибка");
        return 0;
    }
    finally
    {
        Console.WriteLine("Я блок finally");
    }
}
```

```
Console.WriteLine(Divide(4,2));
Console.WriteLine("-----");
Console.WriteLine(Divide(4, 0));
```



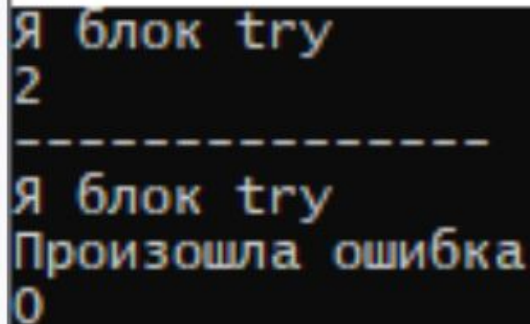
```
Я блок try
Я блок finally
2
-----
Я блок try
Произошла ошибка
Я блок finally
0
```

finally

```
double Divide(int a, int b)
{
    try
    {
        Console.WriteLine("Я блок try");
        return a / b;
    }
    catch (Exception)
    {
        Console.WriteLine("Произошла ошибка");
        return 0;
    }

    Console.WriteLine("Я блок finally");
}
```

```
Console.WriteLine(Divide(4,2));
Console.WriteLine("-----");
Console.WriteLine(Divide(4, 0));
```



```
Я блок try
2
-----
Я блок try
Произошла ошибка
0
```

AppDomain.CurrentDomain.UnhandledException

```
AppDomain.CurrentDomain.UnhandledException += ExceptionHandler;
```

```
void ExceptionHandler(object sender, UnhandledExceptionEventArgs e)
{
    Console.WriteLine("Я завершился с ошибкой");
}
```


LIVE

Порядок перехвата исключений

Порядок перехвата исключений

```
// Исключение-болезнь  
class IllnessException : Exception { }  
  
// Микробная болезнь  
class MicrobeException : IllnessException { }  
  
// Вирусная болезнь  
class VirusException : IllnessException { }
```

Порядок перехвата исключений

```
static void DemoCure()
{
    try
    {
        Live();
    }
    catch (VirusException)
    {
        // тут ловим только VirusException
    }
    catch (IllnessException)
    {
        // тут ловим IllnessException и производные,
        // в т.ч. MicrobeException
        // НО НЕ VirusException
    }
    catch (Exception)
    {
        // тут ловим все остальные исключения
    }
}
```

LIVE

Порядок перехвата исключений

```
try
{
    try
    {
        int.Parse(Console.ReadLine());
    }
    catch (FormatException e)
    {
        Console.WriteLine($"Внутри: {e.Message}");
    }
}
catch (Exception e)
{
    Console.WriteLine($"Снаружи: {e.Message}");
}
```

Порядок перехвата исключений (условные)

```
int GetItem(int[] arr, int index)
{
    try
    {
        return arr[index];
    }
    catch (IndexOutOfRangeException) when (index < 0) // IndexOutOfRangeException для index < 0
    {
        Console.WriteLine("Индекс меньше нуля");
    }
    catch (IndexOutOfRangeException) // IndexOutOfRangeException в остальных случаях
    {
        Console.WriteLine("Индекс аут оф рэндж");
    }
    catch // остальные ошибки
    {
        Console.WriteLine("Другая ошибка");
    }
    return 0;
}
```

LIVE

В чем отличие?

```
static void B()
{
    try
    {
        C();
    }
    catch (Exception e)
    {
        Console.WriteLine("В Ошибка");
        throw e;
    }
}
```

```
static void B()
{
    try
    {
        C();
    }
    catch (Exception e)
    {
        Console.WriteLine("В Ошибка");
        throw;
    }
}
```

LIVE

Что выведется на консоль?

```
try
{
    Console.WriteLine("Я в программе");

    Environment.FailFast("Экстренно падаю");

    Console.WriteLine("Я все еще в программе");
}
catch (Exception e)
{
    Console.WriteLine("Я поймал ошибку");
}
finally
{
    Console.WriteLine("Я в finally");
}
```

LIVE

Best practices

Best practices

- Исключения – для получения источника проблемы
- Создание исключения – дорого, надо пользоваться аккуратно
- Как следствие – если ошибка предсказуема (бизнес данные некорректны) – оформить ошибку без выброса исключения
- Пример: парсинг строки в тип – использовать TryParse, вместо Parse

```
try
{
    var s = int.Parse("s");
    Console.WriteLine(s);
}
catch (FormatException)
{
    Console.WriteLine("Некорректный формат");
}
```

```
if (int.TryParse("s", out var s))
{
    Console.WriteLine(s);
}
else
{
    Console.WriteLine("Некорректный формат");
}
```

LIVE

Best practices

- Важно неповрежденное состояние приложения, а не продуктивность вместо надежности
- Не нужно перехватывать все исключения
- Не должно быть пустых блоков Catch
- Не должно быть блоков Catch для обработки всех исключений без throw
- Логируйте исключения

LIVE

Список материалов для изучения

1. <https://learn.microsoft.com/ru-ru/dotnet/standard/exceptions/>
2. <https://referencesource.microsoft.com/#mscorlib/system/exception.cs.f092fb2b893a0162>
3. <https://otus.ru/journal/vidy-oshibok-programmnogo-obespecheniya-bagi/>
4. <https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/exceptions/>
5. <https://learn.microsoft.com/en-us/dotnet/api/system.appdomain.firstchanceexception?view=net-8.0>
6. <https://learn.microsoft.com/en-us/dotnet/api/system.runtime.compilerservices.runtimewrappedexception?view=net-8.0>
7. <https://learn.microsoft.com/ru-ru/dotnet/standard/exceptions/best-practices-for-exceptions>
8. <https://learn.microsoft.com/ru-ru/dotnet/api/system.exception?view=net-8.0>
9. <https://learn.microsoft.com/en-us/dotnet/standard/exceptions/exception-class-and-properties>
10. <https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.contracts?view=net-8.0>
11. <https://pediaa.com/what-is-the-difference-between-error-and-exception-in-c/>
12. <https://learn.microsoft.com/ru-ru/archive/msdn-magazine/2009/february/clr-inside-out-handling-corrupted-state-exceptions>
13. <https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/february/clr-inside-out-handling-corrupted-state-exceptions>
14. <https://learn.microsoft.com/en-us/aspnet/web-forms/overview/older-versions-getting-started/deploying-web-site-projects/processing-unhandled-exceptions-cs>
15. <https://learn.microsoft.com/en-us/dotnet/framework/configure-apps/file-schema/runtime/legacycorruptedstateexceptionspolicy-element>
16. <https://learn.microsoft.com/ru-ru/dotnet/fundamentals/code-analysis/quality-rules/ca2153>
17. <https://learn.microsoft.com/en-us/windows/win32/seccrypto/common-hresult-values>
18. <https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/finalizers>
19. <https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/statements/using>
20. <https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/statements/lock>
21. <https://qitlab.com/otus-demo/exceptions>
22. <https://learn.microsoft.com/ru-ru/dotnet/framework/interop/how-to-map-hresults-and-exceptions>
- 23.



Вопросы?



Ставим "+",
если вопросы есть



Ставим "-",
если вопросов нет

Рефлексия

**Заполните, пожалуйста,
опрос о занятии
по ссылке в чате**

Спасибо за внимание!

Приходите на следующие вебинары



Михаил Дмитриев

Ведущий программист НИПК Электрон

Разрабатываю и поддерживаю приложения для работы с радиологическими комплексами

<https://t.me/sf321>

