14. What is the difference between “throw” and “throw ex”?

Brief summary: The difference between “throw” and “throw ex” is that “throw” preserves the stack trace (the stack trace will point to the method that caused the exception in the first place) while ”throw ex” does not preserve the stack trace (we will lose the information about the method that caused the exception in the first place. It will seem like the exception was thrown from the place of its catching and re-throwing)

When catching an exception we don’t always want to handle it and then let the program execution continue. Sometimes we want the exception to move on and perhaps be caught in the next catch clause, but we want to log something or do any other action related to this exception occurrence. Such a thing is called **exception re-throwing**. We can do it by either using “throw” or “throw ex”.

The difference between them is that:

- “throw” **preserves the stack trace** (the stack trace will point to the method that caused the exception in the first place)
- “throw ex” **does not preserve the stack trace** (we will lose the information about the method that caused the exception in the first place. It will seem like the exception was thrown from the place of its catching and re-throwing)

We will see this in the code in a second, but first, let’s be sure we understand what the **stack trace** is. The stack trace is a trace of all methods that have been called, that lead to a particular moment in code. Let’s consider the following code:

```
static void MethodA()
{
    Console.WriteLine("In method A");
    MethodB();
}

static void MethodB()
{
    Console.WriteLine("In method B");
    MethodC();
}

static void MethodC()
{
    Console.WriteLine("In method C");
    Console.WriteLine(Environment.StackTrace);
    Console.WriteLine();
}
```

As you can see MethodA calls MethodB which calls MethodC. In the MethodC we log the current state of the stack trace. Let's see what will be printed to the console if I call MethodA from the Main method of the program:

```
In method A
In method B
In method C
   at System.Environment.get_StackTrace()
   at Program.<<Main>$>g__MethodC|0_4() in C:\Users\marta\source\
repos\50InterviewQuestionsMid\ThrowVsThrowEx\Program.cs:line 70
   at Program.<<Main>$>g__MethodB|0_3() in C:\Users\marta\source\
repos\50InterviewQuestionsMid\ThrowVsThrowEx\Program.cs:line 64
   at Program.<<Main>$>g__MethodA|0_2() in C:\Users\marta\source\
repos\50InterviewQuestionsMid\ThrowVsThrowEx\Program.cs:line 58
   at Program.<Main>$(String[] args) in C:\Users\marta\source\rep
os\50InterviewQuestionsMid\ThrowVsThrowEx\Program.cs:line 1
```

All right. At the top of the stack trace we have the method that has been called most recently, and at the bottom - the one that has been called first. The stack trace stores the information about all method calls that lead to the current moment in the program execution. As you can see the getter of the

Environment.StackTrace property is at the very top because we read the stack trace with this method exactly, so it's the latest to be called. Please note that also the numbers of lines of code where those methods have been called are stored in the stack trace.

Stack trace has many uses, but for us as developers the most important value it brings is that it helps us track where some exceptions happened. Imagine that you have a huge app, and when it throws an exception all it says is "object is null!". That wouldn't be very helpful. We want to know the exact method that caused the problem, and even more - the exact line. This will help us to take a look at the right place, place a breakpoint there, and overall solve the problem easier and faster.

All right. We said that "throw" preserves the stack trace while "throw ex" does not - it's sometimes called "resetting the stack trace". Let's see the code that will show us the difference:

```
void MethodThrow()
{
    try
    {
        var collection = Enumerable.Empty<int>();
        var first = collection.First();
    }
    catch (Exception ex)
    {
        throw;
    }
}
```

Here is the first method. As you can see it's designed to throw an exception (it will try to access the first number of an empty collection). And here is the second - almost the same, but doing "throw ex" instead of throw:

```
void MethodThrowEx()
{
    try
    {
        var collection = Enumerable.Empty<int>();
        var first = collection.First();
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

CA2200: Re-throwing caught exception changes stack information

Show potential fixes (Ctrl+.)

As you can see Visual Studio underlines “throw ex” and it has good reasons to do so. We will go back to it in a while. First, let’s see those methods in use. I will call both similarly, so below I’m showing the code for MethodThrow only for brevity.

```
try
{
    MethodThrow();
}
catch (Exception ex)
{
    Console.WriteLine("Throw");
    Console.WriteLine(
        "Exception caught, logging some" +
        " information. Stack trace:\n");
    Console.WriteLine(ex.StackTrace);
    Console.WriteLine();
}
```

All right. Let’s see the output of the program.

```
Throw
Exception caught, logging some information. Stack trace:

   at System.Linq.ThrowHelper.ThrowNoElementsException()
   at System.Linq.Enumerable.First[TSource](IEnumerable`1
 source)
   at Program.<<Main>$>g__MethodThrow|0_0() in C:\Users\m
arta\source\repos\50InterviewQuestionsMid\ThrowVsThrowEx\
Program.cs:line 37
   at Program.<Main>$(String[] args) in C:\Users\marta\so
urce\repos\50InterviewQuestionsMid\ThrowVsThrowEx\Program
.cs:line 5

Throw ex
Exception caught, logging some information. Stack trace:

   at Program.<<Main>$>g__MethodThrowEx|0_1() in C:\Users
\marta\source\repos\50InterviewQuestionsMid\ThrowVsThrowE
x\Program.cs:line 54
   at Program.<Main>$(String[] args) in C:\Users\marta\so
urce\repos\50InterviewQuestionsMid\ThrowVsThrowEx\Program
.cs:line 19
```

For throw, the stack trace ends at Linq.ThrowHelper.ThrowNoElementsException. The previous entry says about the First method, which already gives us some information about the nature of the problem.

For throw ex, the stack trace ends at MethodThrowEx line 54, which is exactly this line:

```
45  void MethodThrowEx()
46  {
47      try
48      {
49          var collection = Enumerable.Empty<int>();
50          var first = collection.First();
51      }
52      catch (Exception ex)
53      {
54          throw ex;
55      }
56  }
```

This means that all information that has been stored in the stack trace before reaching the “throw ex” command is lost. This is not good for us, as we lose

valuable data about the origins of the exception. This is why earlier we saw that Visual Studio suggested to us that using “throw ex” is not a very good idea. **We should stick to using “throw”, not “throw ex”.**

One may wonder “So why is there a possibility to do it in C# at all if we should not use it?”. Well, remember that “ex” is just an object of the Exception class. We throw objects belonging to this class all the time and it’s perfectly fine, only that we throw brand-new exceptions, not the ones that have been already thrown:

```
decimal Divide(decimal a, decimal b)
{
    if(b == 0)
    {
        throw new ArgumentException("B can't be zero!");
    }
    return a / b;
}
```

If the compiler allows us to use “throw (some exception here, should be brand-new)” it can’t really prevent us from throwing an already-thrown exception with “throw ex” even if it’s not a good idea.

Let’s summarize. The difference between “throw” and “throw ex” is that “throw” preserves the stack trace (the stack trace will point to the method that caused the exception in the first place) while ”throw ex” does not preserve the stack trace (we will lose the information about the method that caused the exception in the first place. It will seem like the exception was thrown from the place of its catching and re-throwing).

Bonus questions:

- "**What is the stack trace?**"
  The stack trace is a trace of all methods that have been called, that lead to the current moment of the execution. At the top of the stack trace we have the method that has been called most recently, and at the bottom - the one that has been called first. Stack trace allows us to locate the exact line in code that was the source of an exception.

- "**Should we use “throw” or “throw ex”, and why?**"
  We should use “throw” as it preserves the stack trace and helps us find the original source of the problem.