# Unreachable code, but reachable with an exception

Asked 5 years, 9 months ago  Modified 3 years, 11 months ago  Viewed 18k times

109

This code is part of an application that reads from and writes to an ODBC connected database. It creates a record in the database and then checks if a record has been successfully created, then returning `true`.

My understanding of control flow is as follows:

`command.ExecuteNonQuery()` is documented to throw an `InvalidOperationException` when "a method call is invalid for the object's current state". Therefore, if that would happen, execution of the `try` block would stop, the `finally` block would be executed, then would execute the `return false;` at the bottom.

However, my IDE claims that the `return false;` is unreachable code. And it seems to be true, I can remove it and it compiles without any complaints. However, for me it looks as if there would be no return value for the code path where the mentioned exception is thrown.

```
private static bool createRecord(String table,
                                 IDictionary<String,String> data,
                                 System.Data.IDbConnection conn,
                                 OdbcTransaction trans) {

    [... some other code ...]

    int returnValue = 0;
    try {
        command.CommandText = sb.ToString();
        returnValue = command.ExecuteNonQuery();

        return returnValue == 1;
    } finally {
        command.Dispose();
    }

    return false;
}
```

What is my error of understanding here?

c# exception unreachable-code

Share

Improve this question

edited Mar 23, 2019 at 15:18

Peter Mortensen

31.6k ● 22 ● 109 ● 133

asked Mar 12, 2019 at 8:17

0xCAFEBABE

5,666 ● 5 ● 39 ● 59

Follow

4 learn.microsoft.com/en-us/dotnet/csharp/language-reference/… – Anton Z Mar 12, 2019 at 8:21

43 Side note: do not call `Dispose` explicitly, but put `using`: `using (var command = ...) {command.CommandText = sb.ToString(); return command.ExecuteNonQuery(); }` – Dmitry Bychenko Mar 12, 2019 at 8:22

7 A `finally` block means something else than you think. – Thorbjørn Ravn Andersen Mar 14, 2019 at 9:13

## 9 Answers

Sorted by: Highest score (default)

146

[Compiler Warning (level 2) CS0162]

> Unreachable code detected
>
> The compiler detected code that will never be executed.

Which is just saying, the *Compiler* understands enough through *Static Analysis* that it can't be reached and completely omits it from the compiled IL (hence your warning).

***Note** : You can prove this fact to your self by trying to Step on to the Unreachable Code with the debugger, or using an IL Explorer.*

The `finally` may run on an *Exception*, (though that aside) it doesn't change the fact (in this case) it will still be an *Uncaught Exception*. Ergo, the last `return` will never get hit regardless.

- If you want the code to continue onto the last `return`, your only option is to *Catch* the *Exception*;
- If you don't, just leave it the way it is and remove the `return`.

**Example**

```
try
{
    command.CommandText = sb.ToString();
    returnValue = command.ExecuteNonQuery();

    return returnValue == 1;
}
catch(<some exception>)
{
   // do something
}
```

```
finally
{
    command.Dispose();
}

return false;
```

---

**To quote the documentation**

[try-finally (C# Reference)]

> By using a finally block, you can clean up any resources that are allocated in a try block, and you can run code even if an exception occurs in the try block. Typically, the statements of a finally block run when control leaves a try statement. The transfer of control can occur as a result of normal execution, of execution of a break, continue, goto, or return statement, or of propagation of an exception out of the try statement.
>
> Within a handled exception, the associated finally block is guaranteed to be run. **However, if the exception is unhandled, execution of the finally block is dependent on how the exception unwind operation is triggered. That, in turn, is dependent on how your computer is set up.**
>
> Usually, when an unhandled exception ends an application, whether or not the finally block is run is not important. **However, if you have statements in a finally block that must be run even in that situation, one solution is to add a catch block to the try-finally statement**. Alternatively, **you can catch the exception that might be thrown in the try block of a try-finally statement higher up the call stack**. That is, you can catch the exception in the method that calls the method that contains the try-finally statement, or in the method that calls that method, or in any method in the call stack. If the exception is not caught, execution of the finally block depends on whether the operating system chooses to trigger an exception unwind operation.

**Lastly**

When using anything that supports the `IDisposable` interface (which is designed to release unmanaged resources), you can wrap it in a `using` statement. The compiler will generate a `try {} finally {}` and internally call `Dispose()` on the object.



edited Dec 22, 2020 at 16:05

halfer

20.4k ● 19 ● 108 ● 200

answered Mar 12, 2019 at 8:20

TheGeneral

81.4k ● 9 ● 109 ● 149

1 What do you mean by IL in the first sentences? – Clockwork Mar 13, 2019 at 20:02

2 @Clockwork *IL* is a product of compilation of code written in high-level .NET languages. Once you compile your code written in one of these languages, you will get a binary that is made out of IL. Note that Intermediate Language is sometimes also called Common Intermediate Language (CIL) or Microsoft Intermediate Language (MSIL)., – TheGeneral Mar 13, 2019 at 21:31

1 In short terms, because he didn't catch the possibilities are: Either the try runs until it hits return and thus ignores the return below finally OR an exception is thrown and that return is never reached because the function will exit due to an exception being thrown. – Felype Mar 27, 2019 at 19:49

86

> the finally block would be executed, then would execute the return false; at the bottom.

Wrong. `finally` doesn't swallow the exception. It honors it and the exception will be thrown as normal. It will only execute the code in the finally before the block ends (with or without an exception).

If you want the exception to be swallowed, you should use a `catch` block with no `throw` in it.

Share Improve this answer Follow

answered Mar 12, 2019 at 8:19

Patrick Hofman
**157k** ● 23 ● 263 ● 334

1 will the above sinppet compile in case exception, what will be returned? – Ehsan Sajjad Mar 12, 2019 at 8:31

3 It does compile, but it will never hit `return false` since it will throw an exception instead @EhsanSajjad – Patrick Hofman Mar 12, 2019 at 8:32

1 seems strange, compiles because either it will return a value for bool in case of no exception and in case of exception nothing will be, so legit for to satisfy method's return type ? – Ehsan Sajjad Mar 12, 2019 at 8:35

2 The compiler will just ignore the line, that is what the warning is for. So why is that strange? @EhsanSajjad – Patrick Hofman Mar 12, 2019 at 8:36

3 Fun fact: It is actually **not** guaranteed that a finally block will run if the exception is not caught in the program. The spec doesn't guarantee this and early CLRs did **NOT** execute the finally block. I think starting with 4.0 (might have been earlier) that behavior changed, but other runtimes might still behave differently. Makes for rather surprising behavior. – Voo Mar 12, 2019 at 20:31

27

The warning is because you didn't use `catch` and your method is basically written like this:

```
bool SomeMethod()
{
    return true;
    return false; // CS0162 Unreachable code detected
}
```

Since you use `finally` solely to dispose, the preferred solution is to utilize `using` pattern:

```
using(var command = new WhateverCommand())
{
     ...
}
```

That's enough, to ensure what `Dispose` will be called. It's guaranteed to be called either after successful execution of code block or upon (before) some `catch` *down* in call stack (parent calls are down, right?).

If it wouldn't be about disposing, then

```
try { ...; return true; } // only one return
finally { ... }
```

is enough, since you will **never** have to return `false` at the end of method (there is no need for that line). Your method is either return result of command execution (`true` or `false`) or will throw an exception *otherwise*.

---

Consider also to throw own exceptions by wrapping expected exceptions (check out InvalidOperationException constructor):

```
try { ... }
catch(SomeExpectedException e)
{
    throw new SomeBetterExceptionWithExplanaition("...", e);
}
```

This is typically used to say something more meaningful (useful) to the caller than nested call exception would be telling.

---

Most of times you don't really care about unhandled exceptions. Sometimes you need to ensure that `finally` is called even if exception is unhandled. In this case you

simply catch it yourself and re-throw (see this answer):

```
try { ... }
catch { ...; throw; } // re-throw
finally { ... }
```

Share

Improve this answer

Follow

edited Mar 21, 2019 at 20:07

marc_s

753k ● 183 ● 1.4k ● 1.5k

answered Mar 12, 2019 at 9:29

Sinatr

21.9k ● 17 ● 103 ● 335

14

It seems, you are looking for something like this:

```
private static bool createRecord(string table,
                                 IDictionary<String,String> data,
                                 System.Data.IDbConnection conn,
                                 OdbcTransaction trans) {
  [... some other code ...]

  // Using: do not call Dispose() explicitly, but wrap IDisposable into using
  using (var command = ...) {
    try {
      // Normal flow:
      command.CommandText = sb.ToString();

      // True if and only if exactly one record affected
      return command.ExecuteNonQuery() == 1;
    }
    catch (DbException) {
      // Exceptional flow (all database exceptions)
      return false;
    }
  }
}
```

Please, note, that `finally` *doesn't swallow* any exception

```
finally {
  // This code will be executed; the exception will be efficently re-thrown
}

// And this code will never be reached
```

Share

Improve this answer

Follow

edited Mar 23, 2019 at 15:20

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Mar 12, 2019 at 8:30

Dmitry Bychenko

186k ● 20 ● 171 ● 225

You don't have a `catch` block, so the exception is still thrown, which blocks the return.

8

> the finally block would be executed, then would execute the return false; at the bottom.

This is wrong, because the finally block would be executed, and then there would be an uncaught exception.

`finally` blocks are used for cleanup, and they do not catch the exception. The exception is thrown before the return, therefore, the return will never be reached, because an exception is thrown before.

Your IDE is correct that it will never be reached, because the exception will be thrown. Only `catch` blocks are able to catch exceptions.

Reading from the documentation,

> Usually, when an unhandled exception ends an application, whether or not the finally block is run is not important. However, if you have statements in a finally block that must be run even in that situation, one solution is to **add a catch block to the try-finally statement**. Alternatively, you can catch the exception that might be thrown in the try block of a try-finally statement higher up the call stack. That is, you can catch the exception in the method that calls the method that contains the try-finally statement, or in the method that calls that method, or in any method in the call stack. **If the exception is not caught, execution of the finally block depends on whether the operating system chooses to trigger an exception unwind operation**.

This clearly shows that the finally is not intended to catch the exception, and you would have been correct if there had been an empty `catch` statement before the `finally` statement.

Share Improve this answer Follow

answered Mar 12, 2019 at 23:45

Junelilly

394 ● 4 ● 16

7

When the exception is thrown, the stack will unwind (execution will move out of the function) without returning a value, and any catch block in the stack frames above the function will catch the exception instead.

Hence, `return false` will never execute.

Try manually throwing an exception to understand the control flow:

```
try {
    command.CommandText = sb.ToString();
    returnValue = command.ExecuteNonQuery();

    // Try this.
    throw new Exception("See where this goes.");

    return returnValue == 1;
} finally {
    command.Dispose();
}
```

Share Improve this answer Follow

answered Mar 12, 2019 at 8:20

Nisarg Shah

14.5k ● 6 ● 37 ● 57

5

On your code:

```
private static bool createRecord(String table, IDictionary<String,String> data,
System.Data.IDbConnection conn, OdbcTransaction trans) {

    [... some other code ...]

    int returnValue = 0;
    try {
        command.CommandText = sb.ToString();
        returnValue = command.ExecuteNonQuery();

        return returnValue == 1; // You return here in case no exception is
thrown
    } finally {
        command.Dispose(); //You don't have a catch so the exception is passed
on if thrown
    }

    return false; // This is never executed because there was either one of the
above two exit points of the method reached.
}
```

> the finally block would be executed, then would execute the return false; at the bottom

This is the flaw in your logic because the `finally` block won't catch the exception and it will never reach the last return statement.

Share Improve this answer Follow

answered Mar 12, 2019 at 8:25

meJustAndrew

6,583 ● 10 ● 55 ● 86

4

The last statement `return false` is unreachable, because the try block is missing a `catch` part that would handle the exception, so the exception is rethrown after the `finally` block and the execution never reaches the last statement.

Share

Improve this answer

Follow

edited Mar 12, 2019 at 8:28

answered Mar 12, 2019 at 8:20

Martin Staufcik

9,440 ● 7 ● 55 ● 74

2

You have two return paths in your code, the second of which is unreachable because of the first. The last statement in your `try` block `return returnValue == 1;` provides your normal return, so you can never reach the `return false;` at the end of the method block.

FWIW, order of exection related to the `finally` block is: the expression supplying the return value in the try block will be evaluated first, then the finally block will be executed, and then the calculated expression value will be returned (inside the try block).

Regarding flow on exception... without a `catch`, the `finally` will be executed upon exception before the exception is then rethrown out of the method; there is no "return" path.

Share

Improve this answer

Follow

edited Mar 13, 2019 at 21:40

answered Mar 13, 2019 at 21:31

C Robinson

459 ● 5 ● 21