The Secret Life of Xcode Breakpoints

Vincent Pradeilles (@v_pradeilles) – Worldline

Breakpoints, really? 🤔

Breakpoints in Xcode actually cover a lot of debugging use cases!

Let’s look at some of them 💪

Use Case #1

“I want to dump some variable to the console”

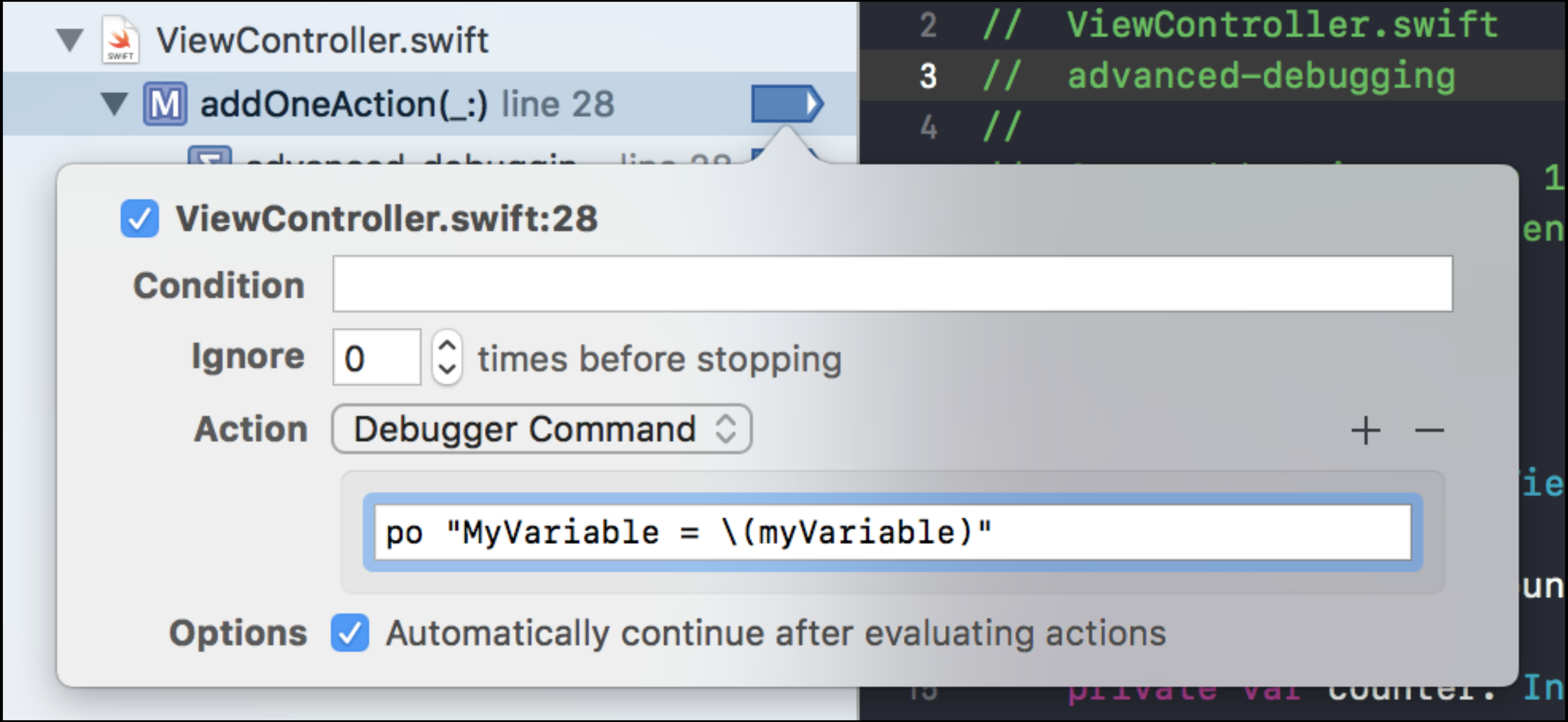
Use Case #1

“I want to dump some variable to the console”

That’s a pretty common use case!

Easy approach: just use `print()`

Drawback: you need to rebuild your project 👎



```
//  ViewController.swift
//  advanced-debugging
//
```

ViewController.swift:28

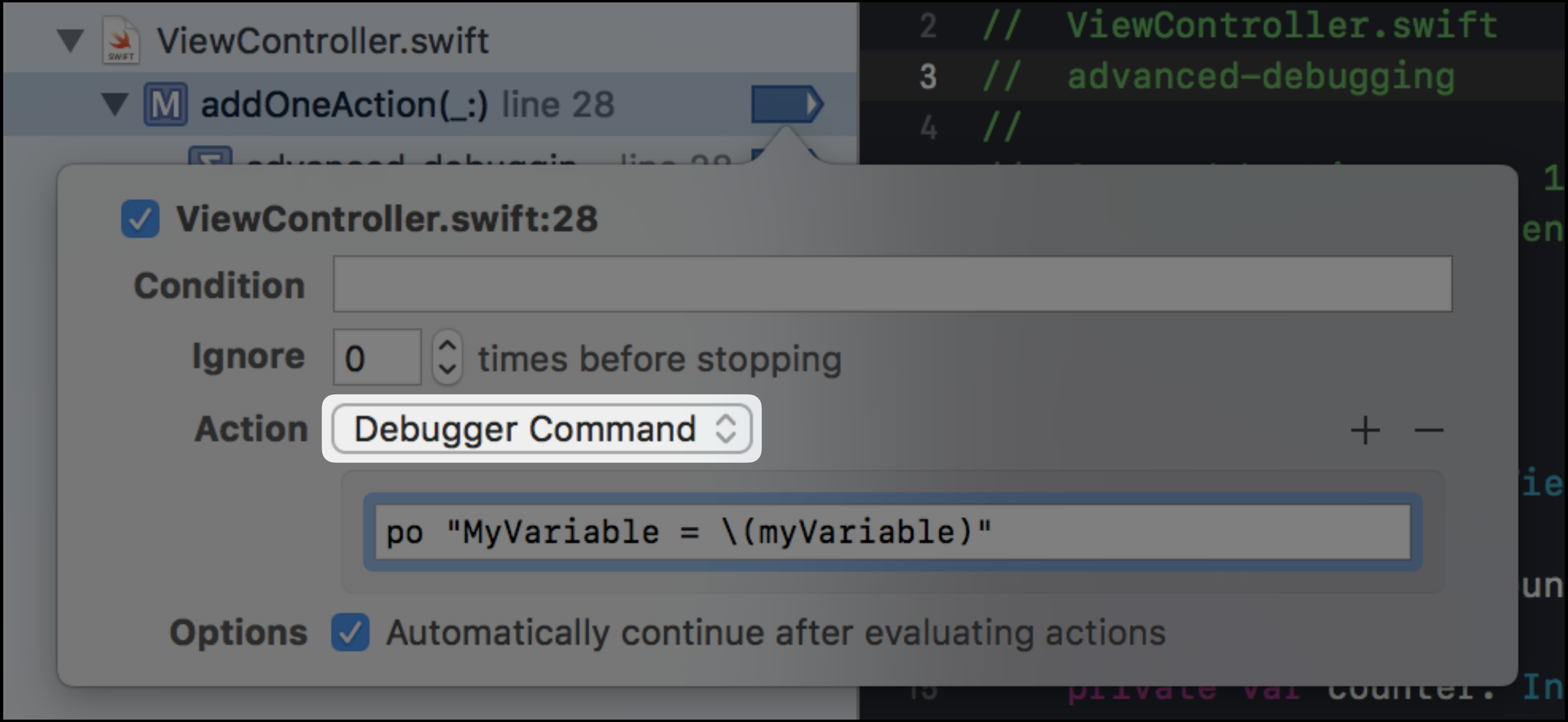
Condition

Ignore 0 times before stopping

Action Debugger Command

```
po "MyVariable = \(myVariable)"
```

Options ✓ Automatically continue after evaluating actions


ViewController.swift

addOneAction(_:) line 28

ViewController.swift:28

Condition

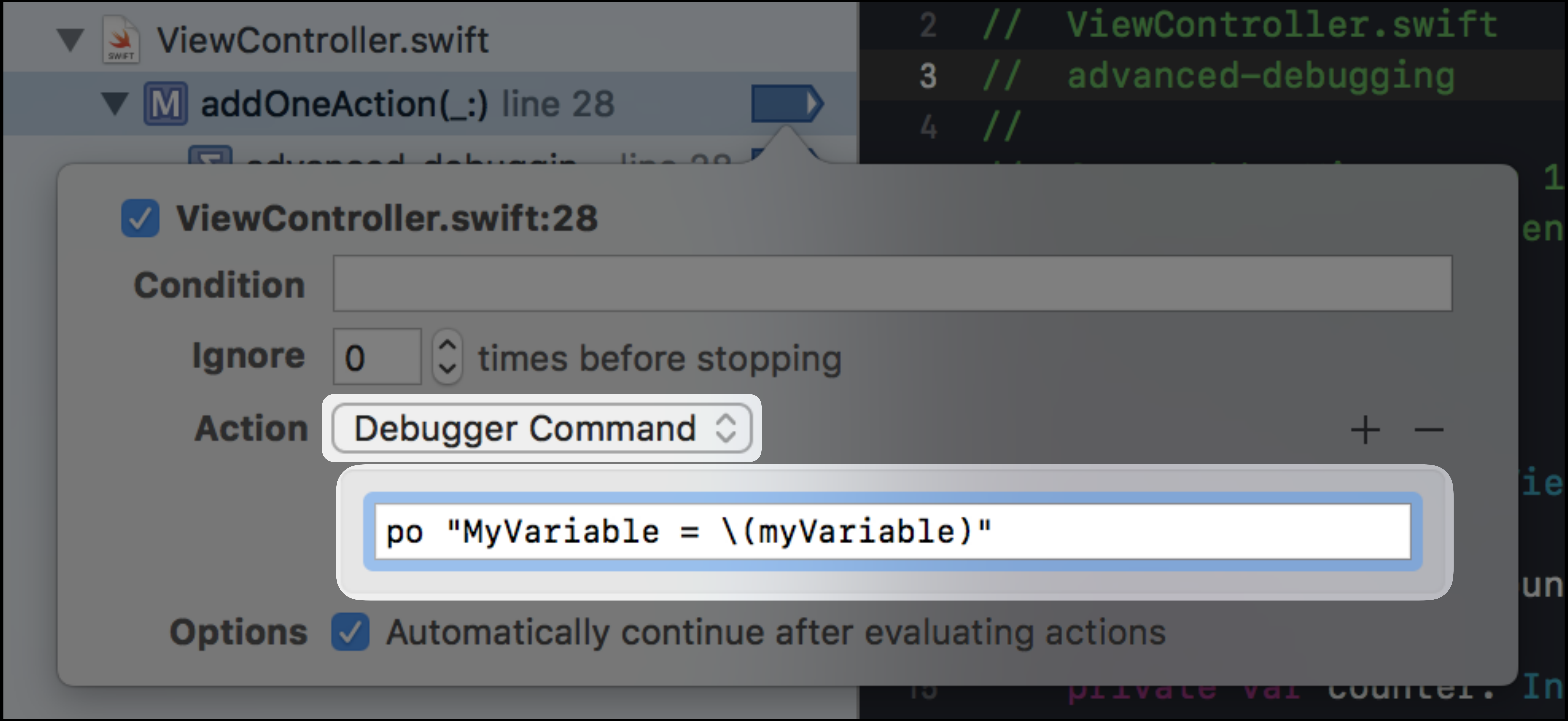
Ignore 0 times before stopping

Action Debugger Command

```
po "MyVariable = \(myVariable)"
```

Options ✓ Automatically continue after evaluating actions

```
//  ViewController.swift
//  advanced-debugging
//
```



ViewController.swift

addOneAction(_:) line 28

```
// ViewController.swift
// advanced-debugging
//
```

ViewController.swift:28

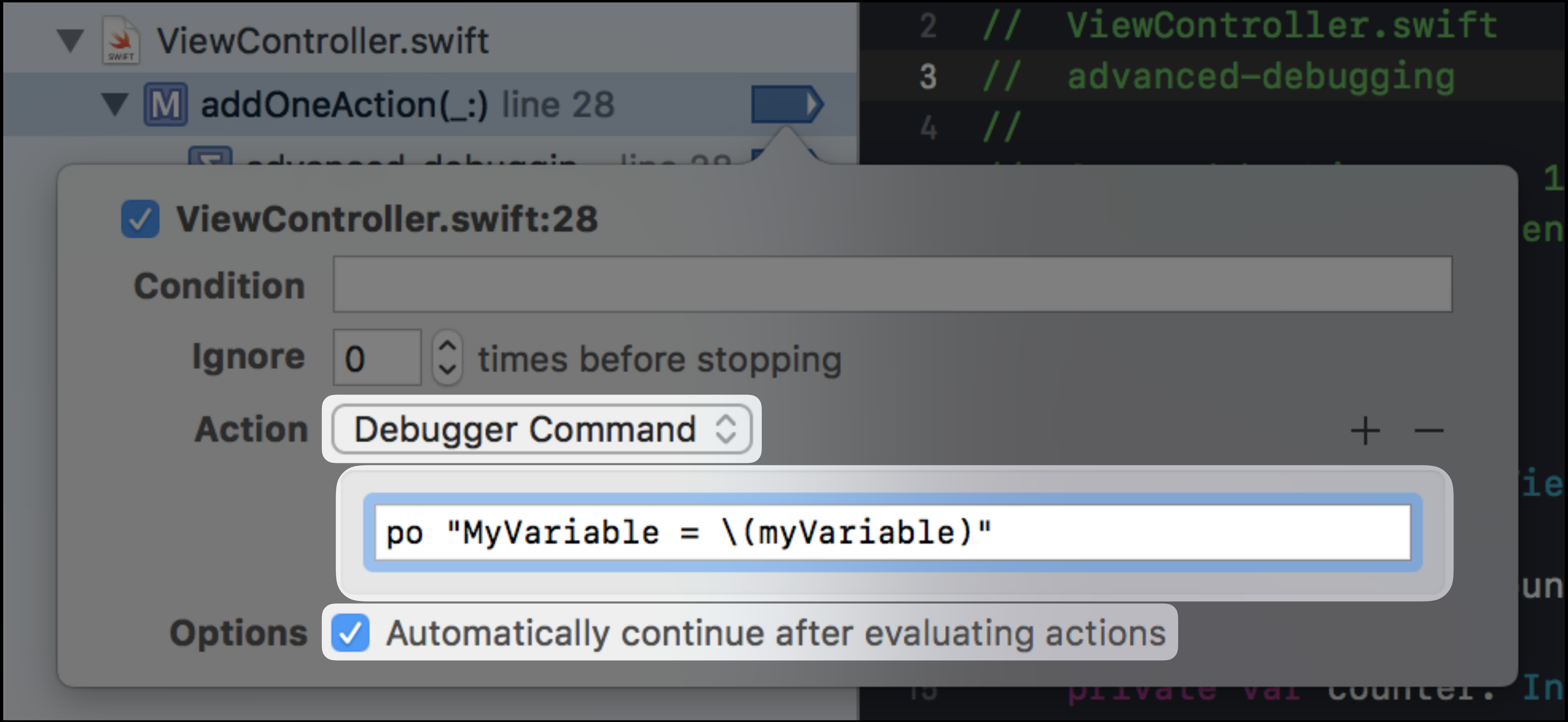
Condition

Ignore 0 times before stopping

Action: Debugger Command

```
po "MyVariable = \(myVariable)"
```

Options: [✓] Automatically continue after evaluating actions



```
//  ViewController.swift
//  advanced-debugging
//
```

☑ ViewController.swift:28

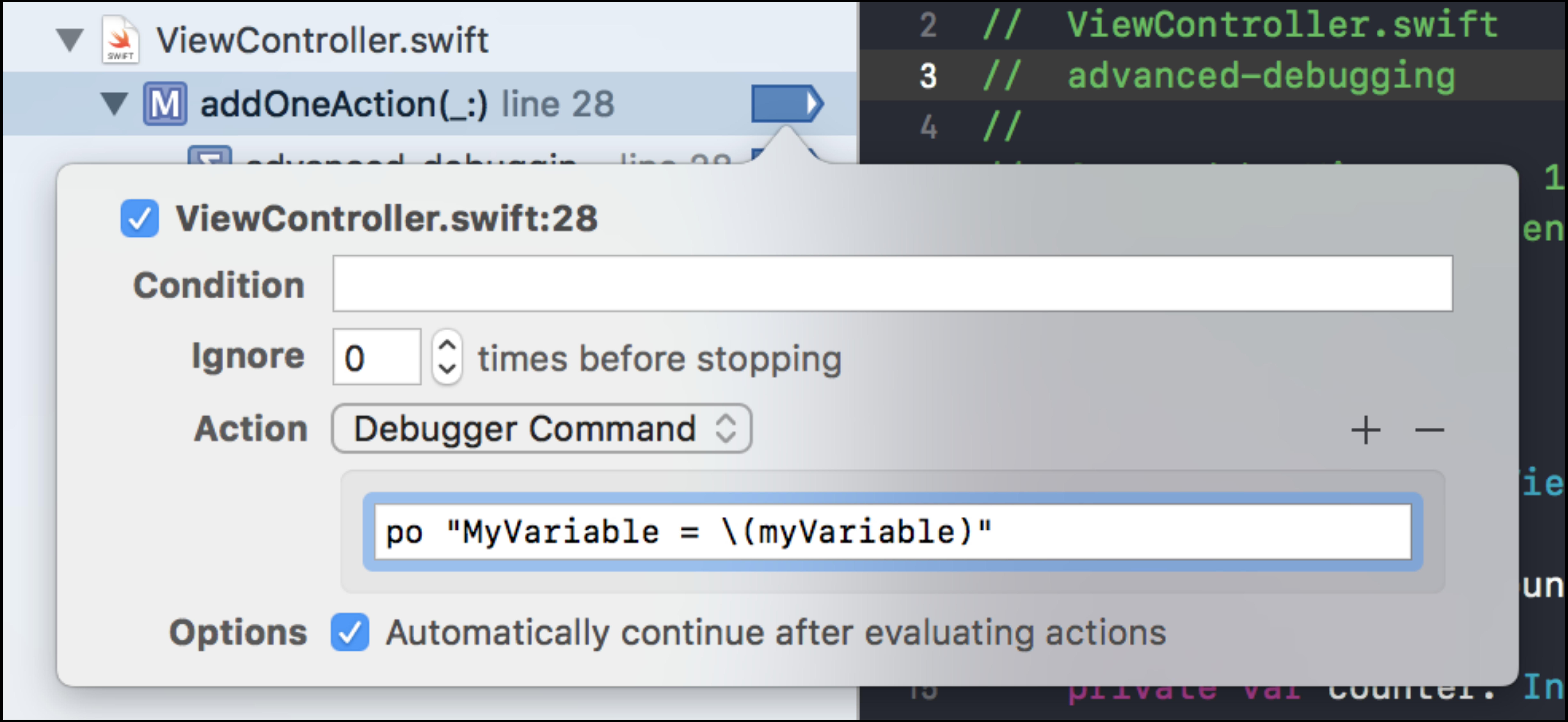
Condition

Ignore 0 times before stopping

Action: Debugger Command

```
po "MyVariable = \(myVariable)"
```

Options: ☑ Automatically continue after evaluating actions



ViewController.swift:28

Condition

Ignore 0 times before stopping

Action Debugger Command

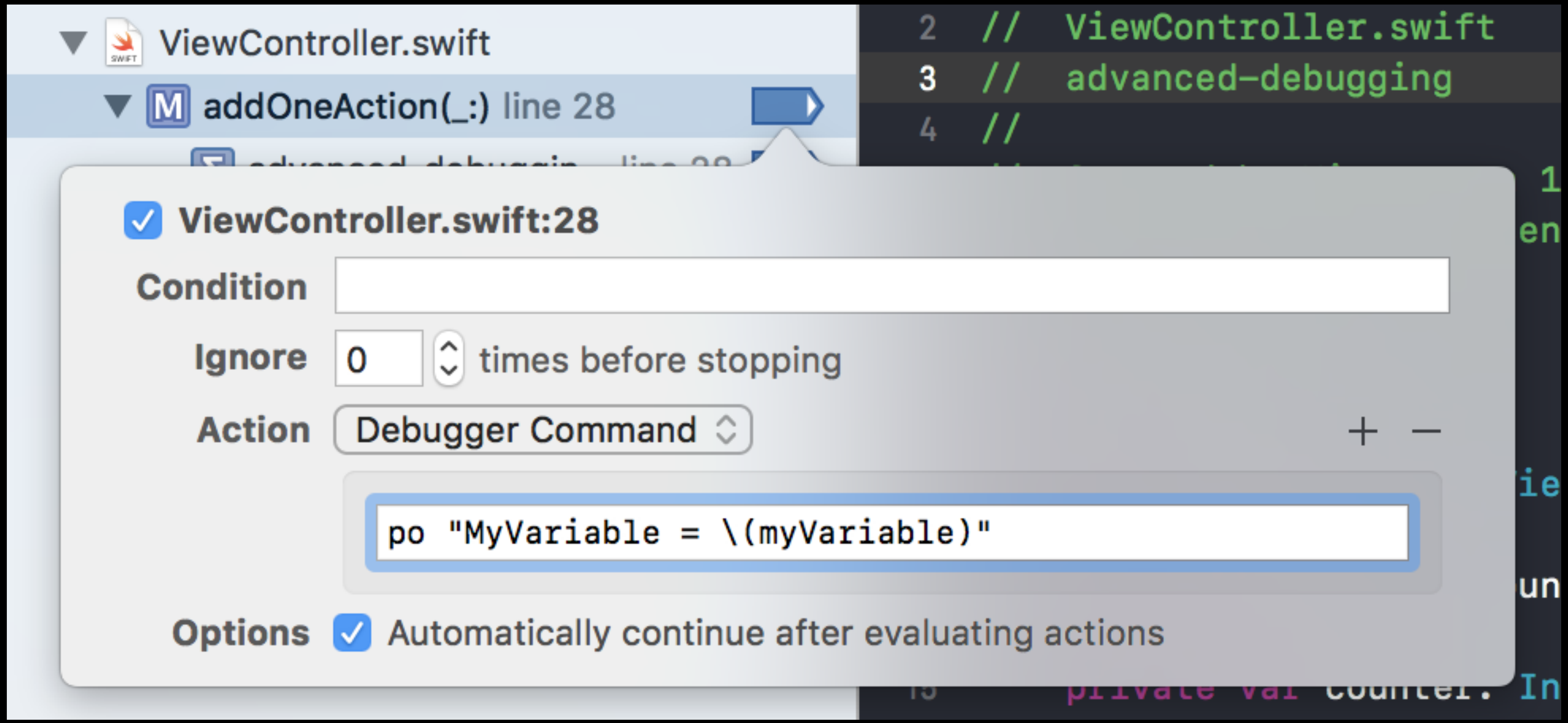
```
po "MyVariable = \(myVariable)"
```

Options ✓ Automatically continue after evaluating actions

```
// ViewController.swift
// advanced-debugging
//
```

No need to rebuild 👍

Maybe even no need to relaunch 🤩



ViewController.swift:28

Condition

Ignore 0 times before stopping

Action Debugger Command

```
po "MyVariable = \(myVariable)"
```

Options ✓ Automatically continue after evaluating actions

Use Case #2

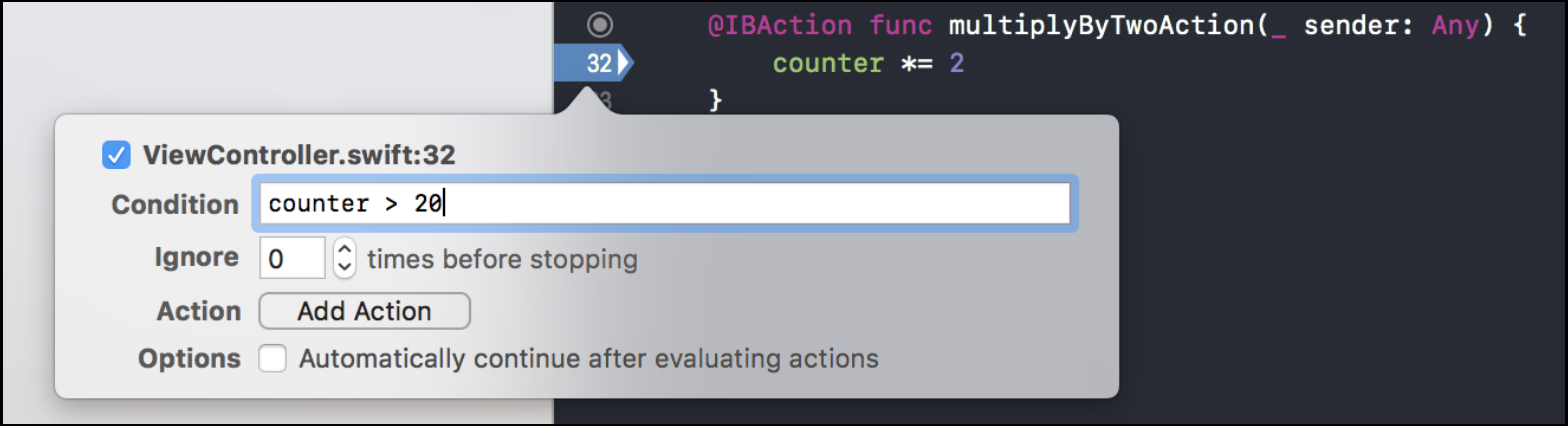
“I don’t always want my breakpoint to trigger”

Use Case #2

“I don’t always want my breakpoint to trigger”

We need to debug code that is heavily called, but we’re only interested in some specific calls.

(Think of low-level networking, ressource loading, etc.)



```
@IBAction func multiplyByTwoAction(_ sender: Any) {
    counter *= 2
}
```

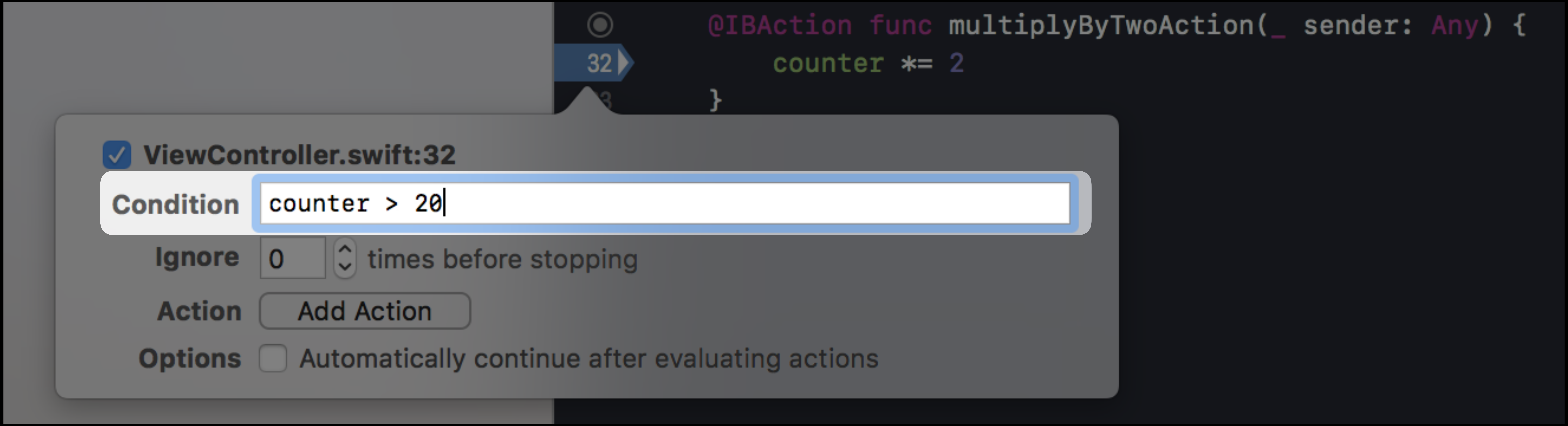
☑ ViewController.swift:32

Condition: `counter > 20`

Ignore: 0 times before stopping

Action: Add Action

Options: ☐ Automatically continue after evaluating actions



```
@IBAction func multiplyByTwoAction(_ sender: Any) {
    counter *= 2
}
```

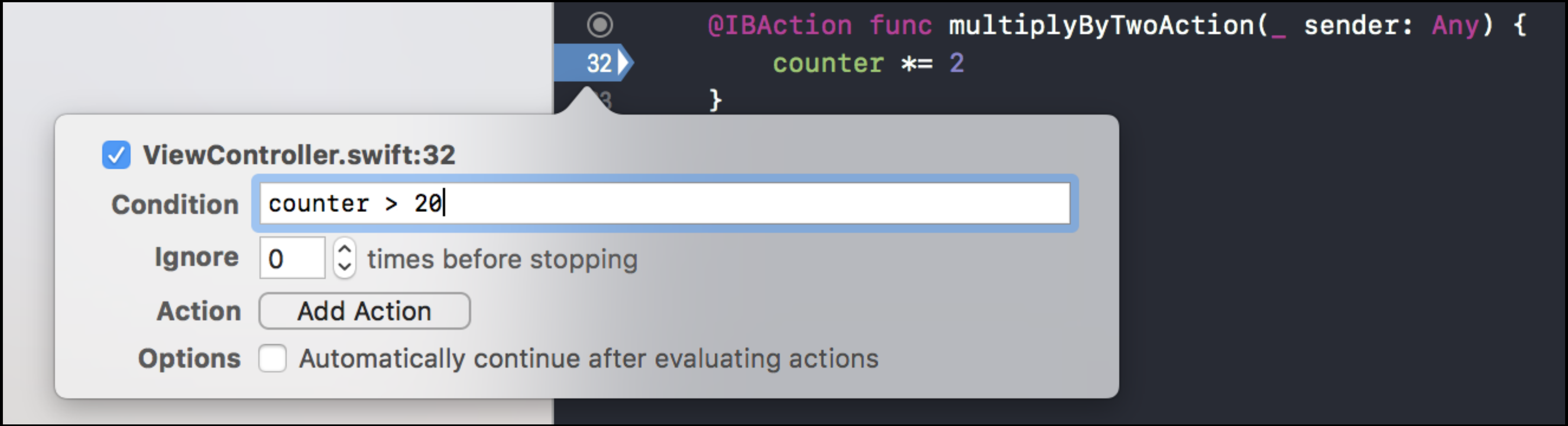
ViewController.swift:32

Condition: counter > 20

Ignore: 0 times before stopping

Action: Add Action

Options: Automatically continue after evaluating actions



```
@IBAction func multiplyByTwoAction(_ sender: Any) {
    counter *= 2
}
```

ViewController.swift:32

Condition: `counter > 20`

Ignore 0 times before stopping

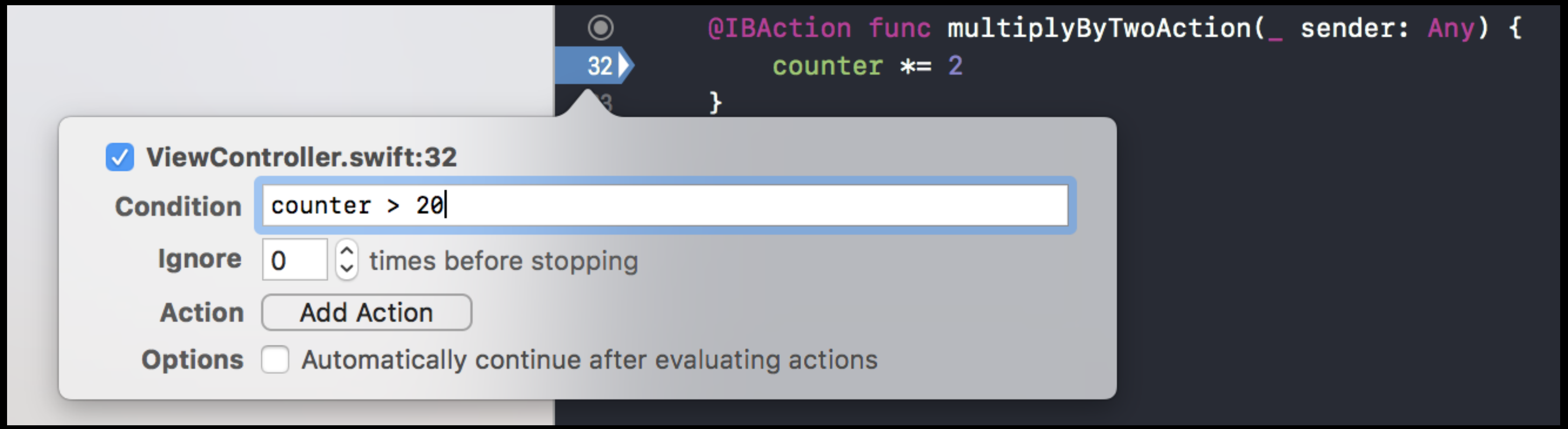
Action: Add Action

Options: Automatically continue after evaluating actions

Our breakpoint will only trigger when we need it to 👍

Boolean operators can be used (&&, ||, !)

Function calls can be made



```
@IBAction func multiplyByTwoAction(_ sender: Any) {
    counter *= 2
}
```

☑ ViewController.swift:32

Condition `counter > 20`

Ignore 0 times before stopping

Action Add Action

Options ☐ Automatically continue after evaluating actions

Use Case #3

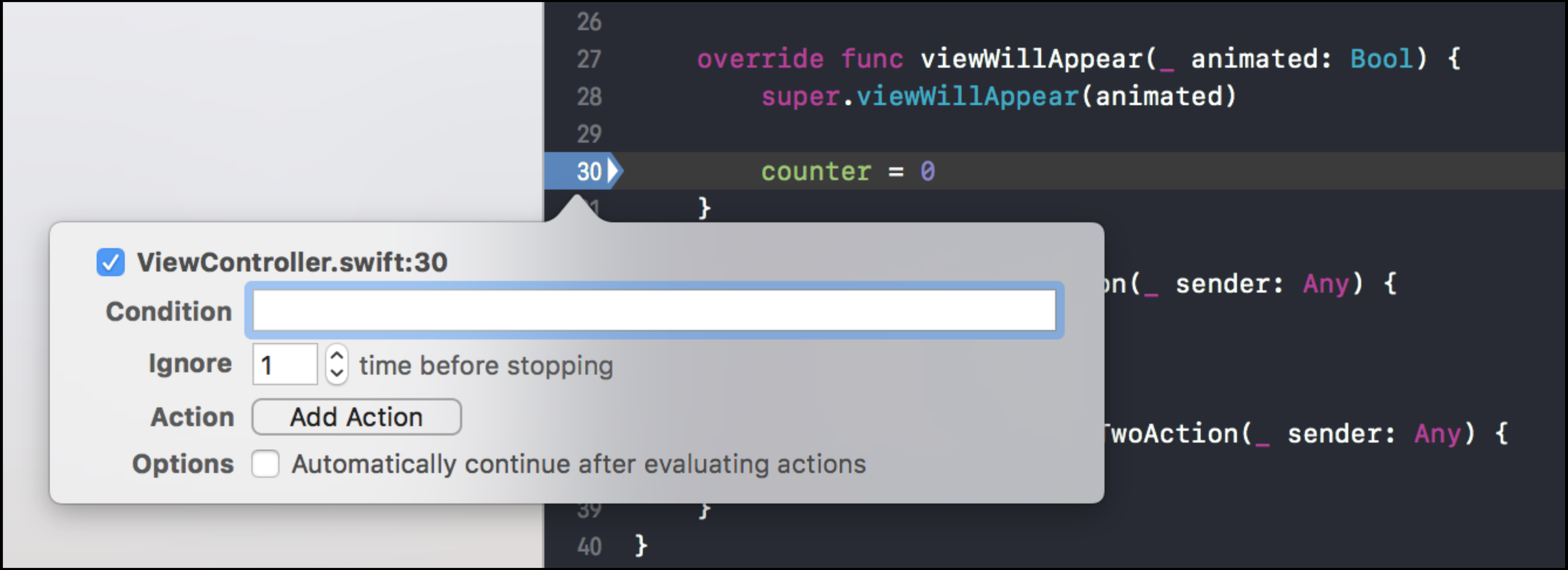
“My break point shouldn’t trigger the first time”

Use Case #3

“My break point shouldn’t trigger the first time”

In a way, that’s a special version of the previous use case

But to implement it with a conditional breakpoint, we’d need to had a variable to our code 👎



```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)

    counter = 0
}
```

```
n(_ sender: Any) {
```

```
TwoAction(_ sender: Any) {
```

```
    }
}
```

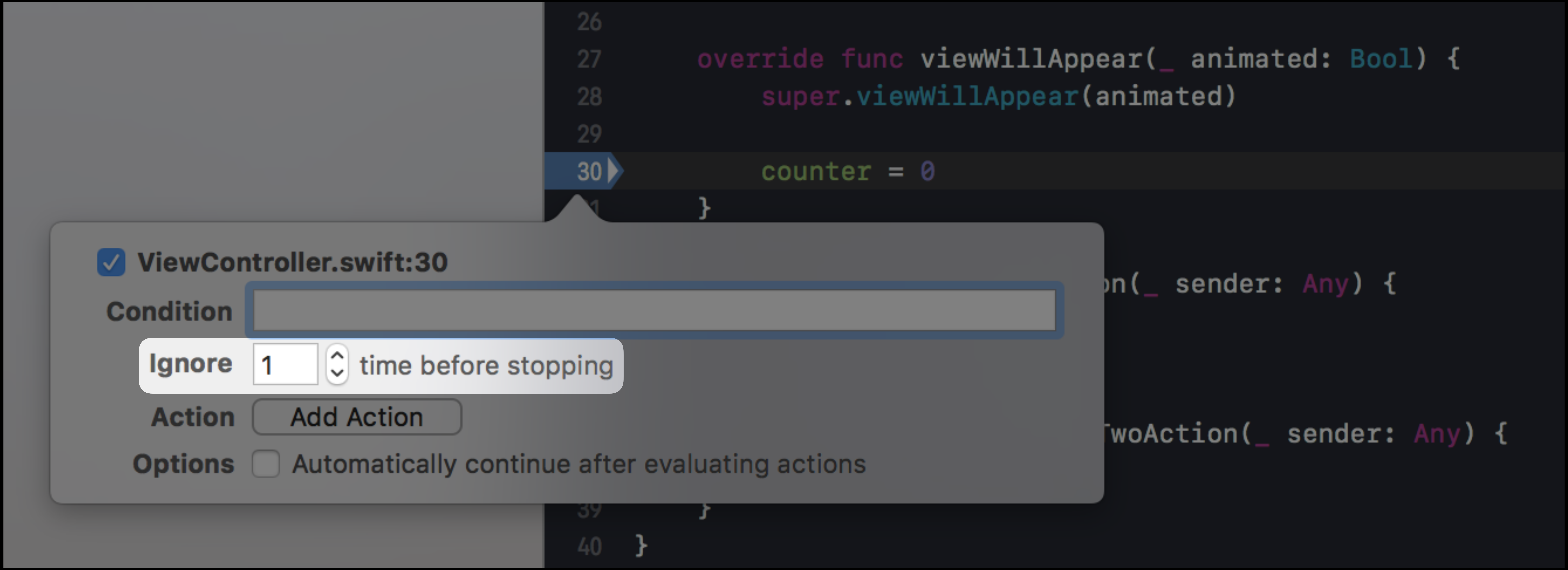
[✓] ViewController.swift:30

Condition:

Ignore: 1 time before stopping

Action: Add Action

Options: [ ] Automatically continue after evaluating actions



```
    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        counter = 0
    }
```

☑ ViewController.swift:30

Condition

Ignore 1 time before stopping

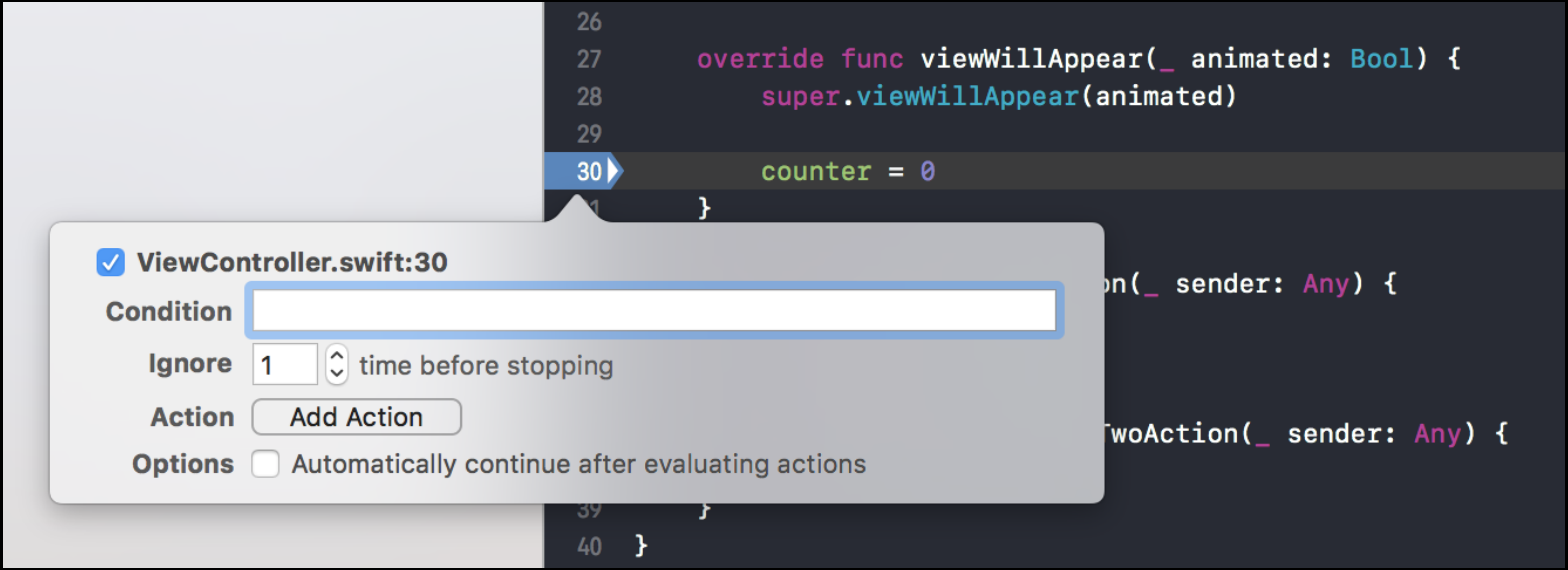
Action Add Action

Options ☐ Automatically continue after evaluating actions

```
n(_ sender: Any) {

TwoAction(_ sender: Any) {

    }
}
```



```
    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        counter = 0
    }
```

```
n(_ sender: Any) {
```

```
TwoAction(_ sender: Any) {
```

```
    }
}
```

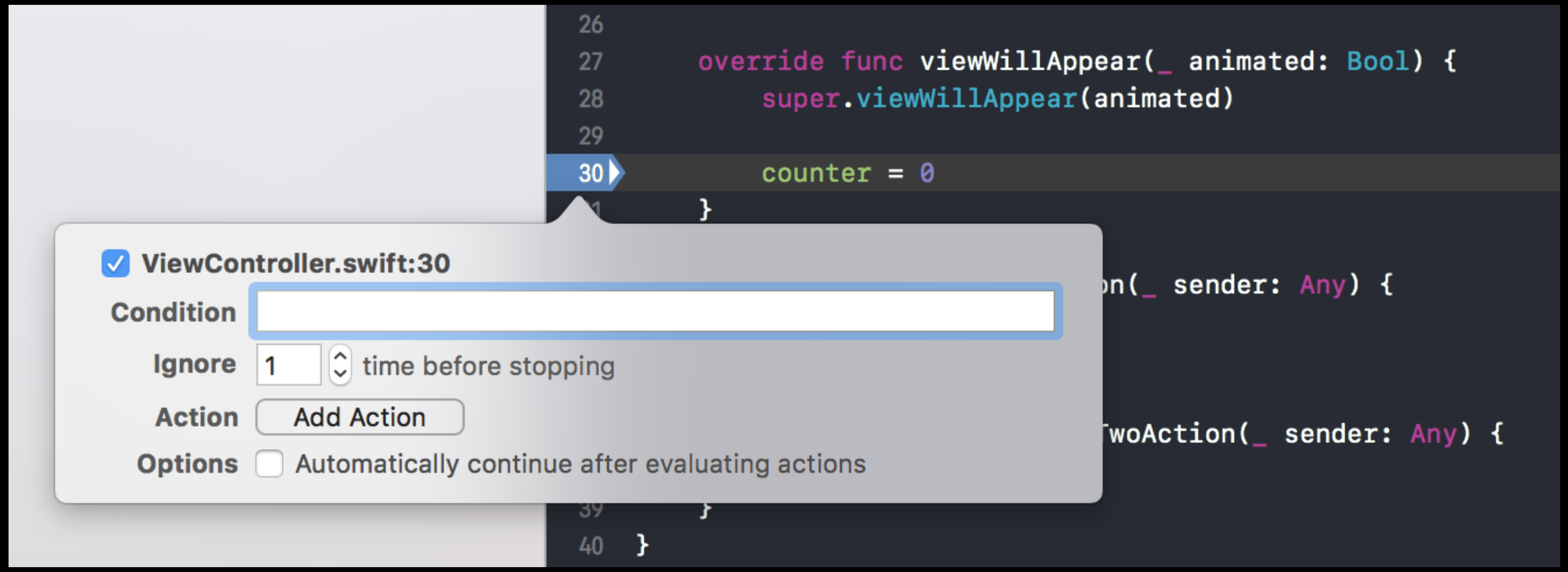
☑ ViewController.swift:30

Condition

Ignore 1 time before stopping

Options ☐ Automatically continue after evaluating actions

Super useful when debugging methods such as `viewWillAppear(_:)` or `viewDidLayoutSubviews()`



```
    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        counter = 0
    }
```

ViewController.swift:30

Condition

Ignore 1 time before stopping

Action Add Action

Options Automatically continue after evaluating actions

Use Case #4

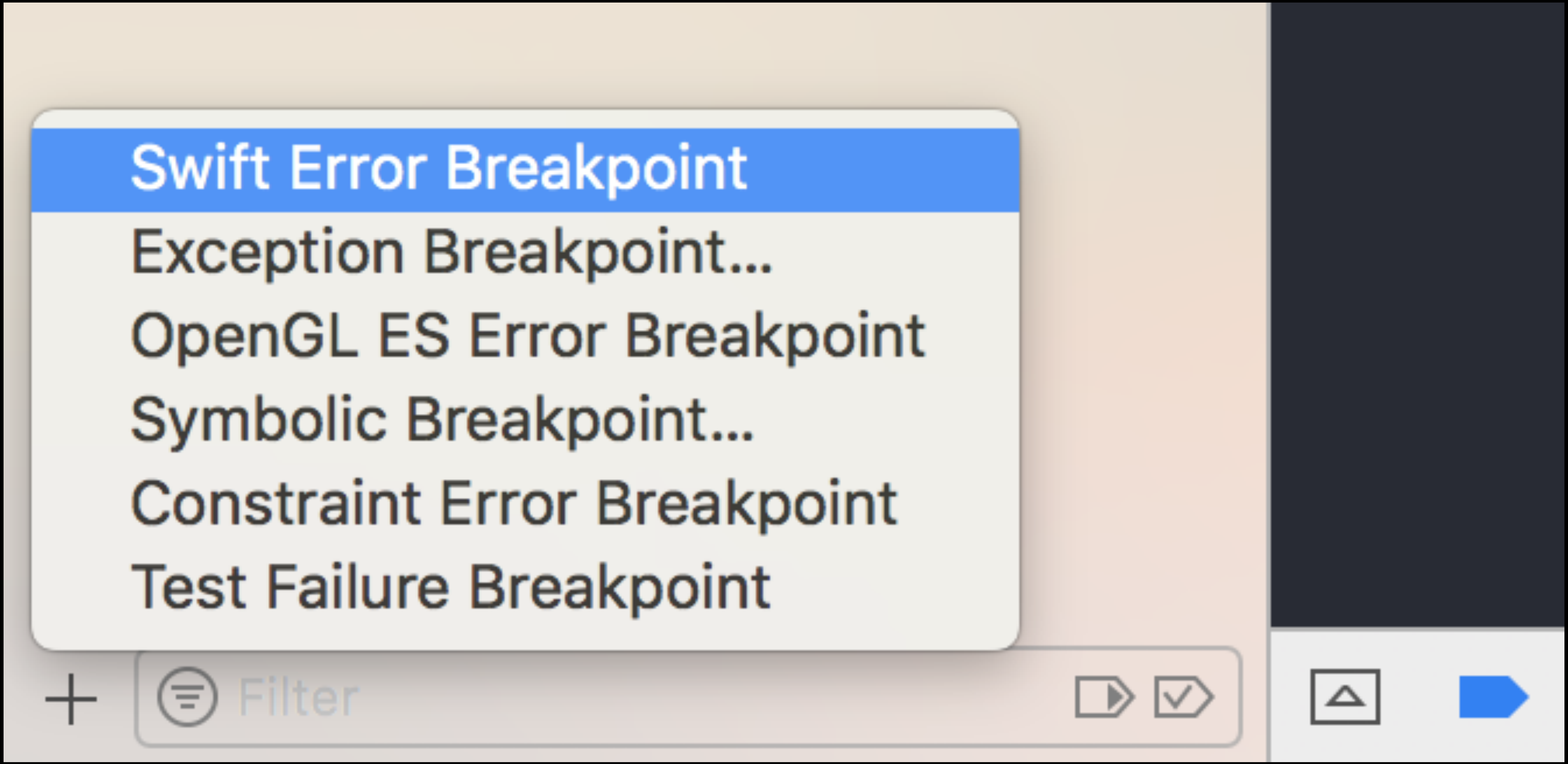
“I want to know which method threw an error”

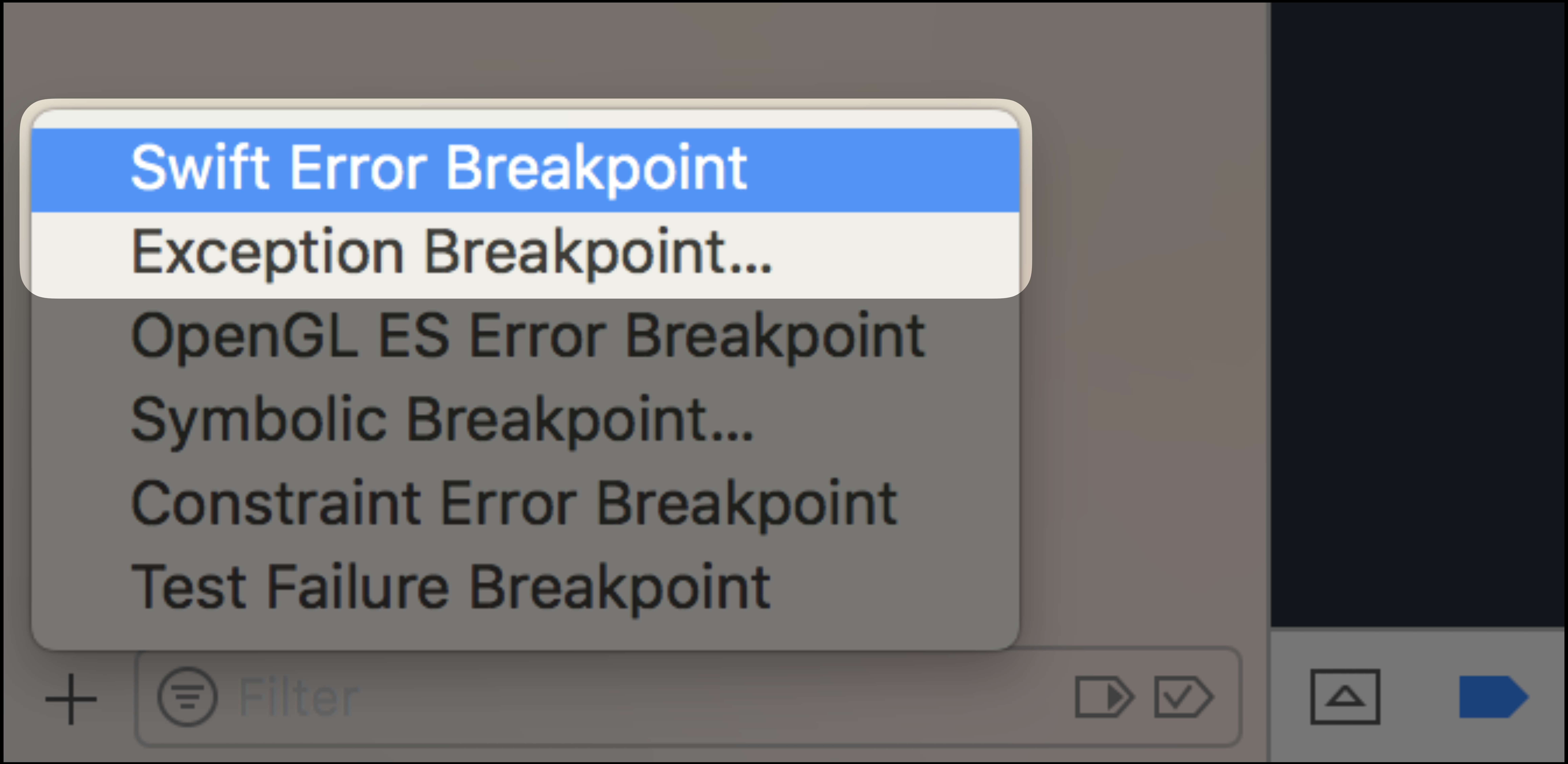
Use Case #4

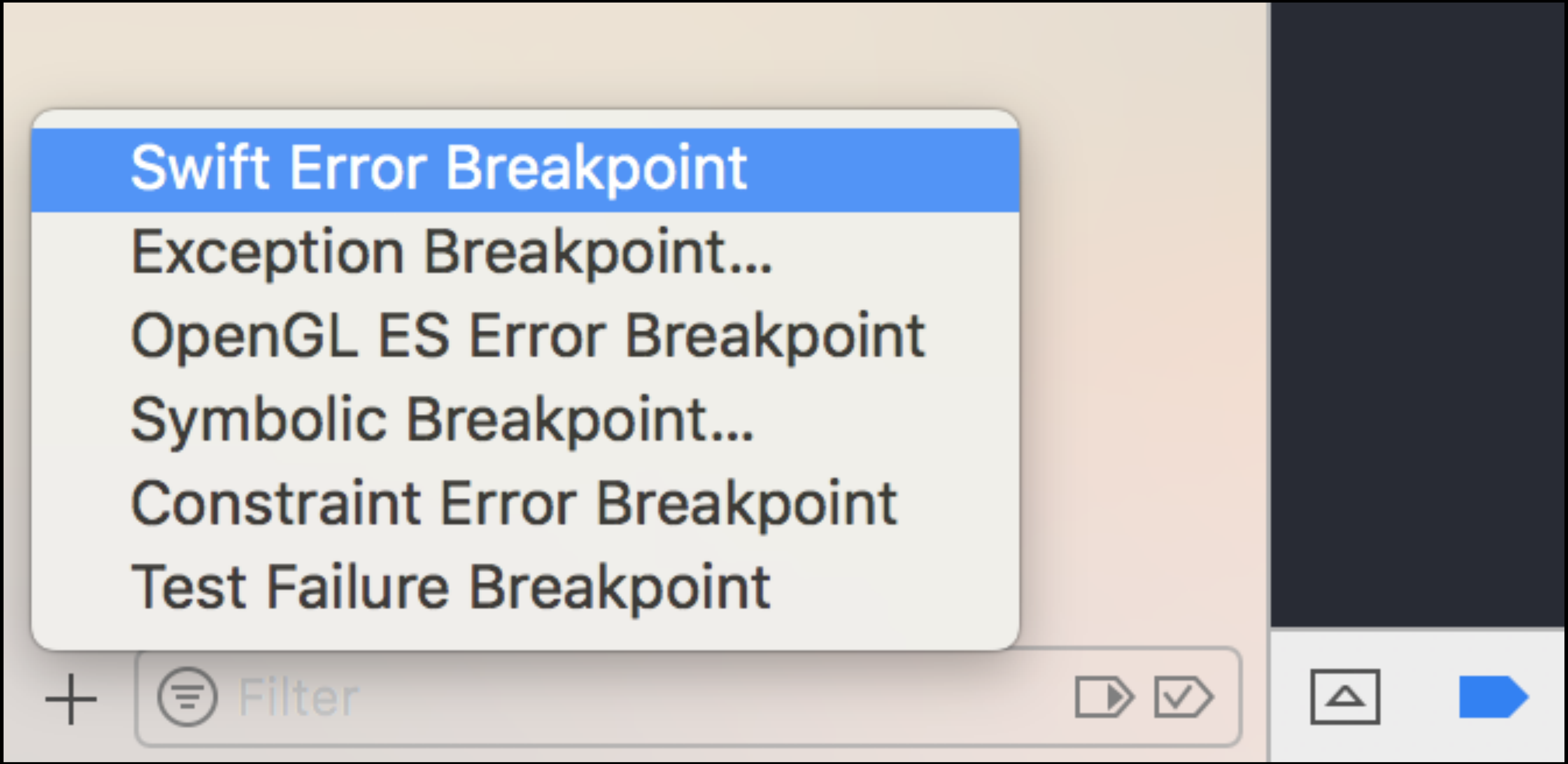
“I want to know which method threw an error”

Think of JSON decoding: when an error happens, we want to pinpoint its origin

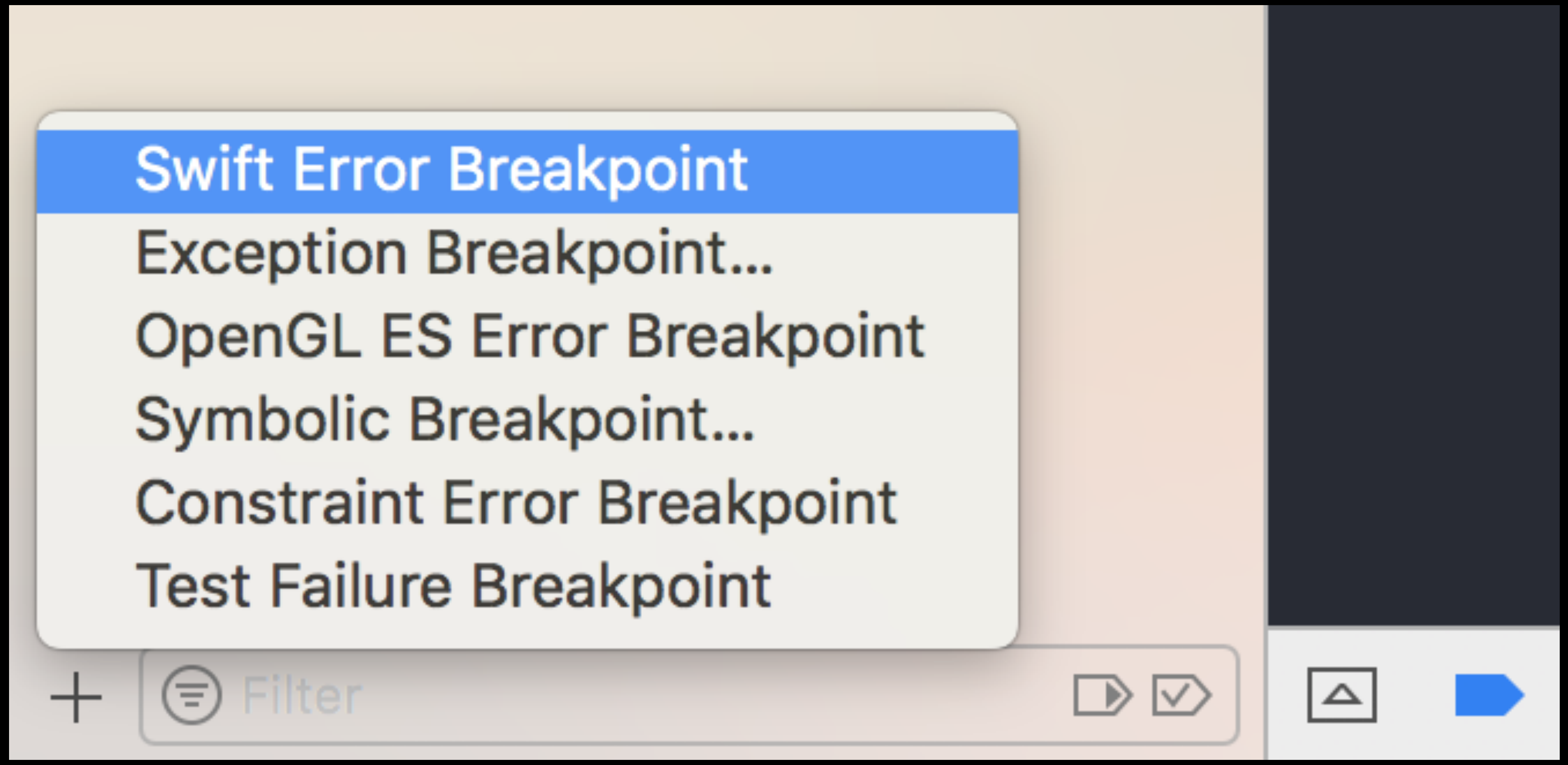
And we’d like to do it fast!


Swift Error Breakpoint
Exception Breakpoint...
OpenGL ES Error Breakpoint
Symbolic Breakpoint...
Constraint Error Breakpoint
Test Failure Breakpoint
Filter


Swift Error Breakpoint
Exception Breakpoint...
OpenGL ES Error Breakpoint
Symbolic Breakpoint...
Constraint Error Breakpoint
Test Failure Breakpoint
+
Filter


Swift Error Breakpoint
Exception Breakpoint...
OpenGL ES Error Breakpoint
Symbolic Breakpoint...
Constraint Error Breakpoint
Test Failure Breakpoint
Filter

Whenever an error or exception happens, you'll now which code was responsible 👌



Use Case #5

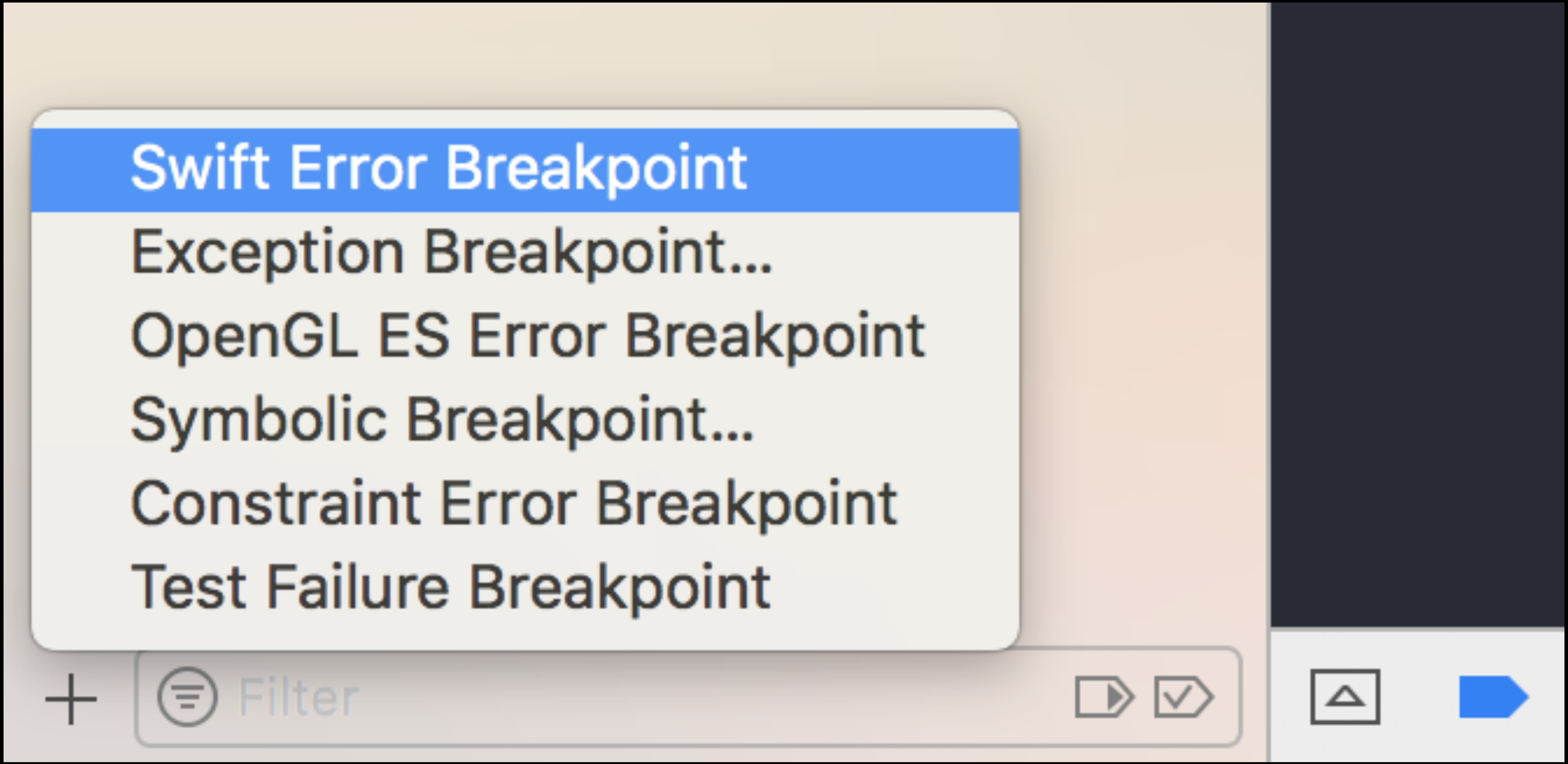
“I want to know when a framework method was called”

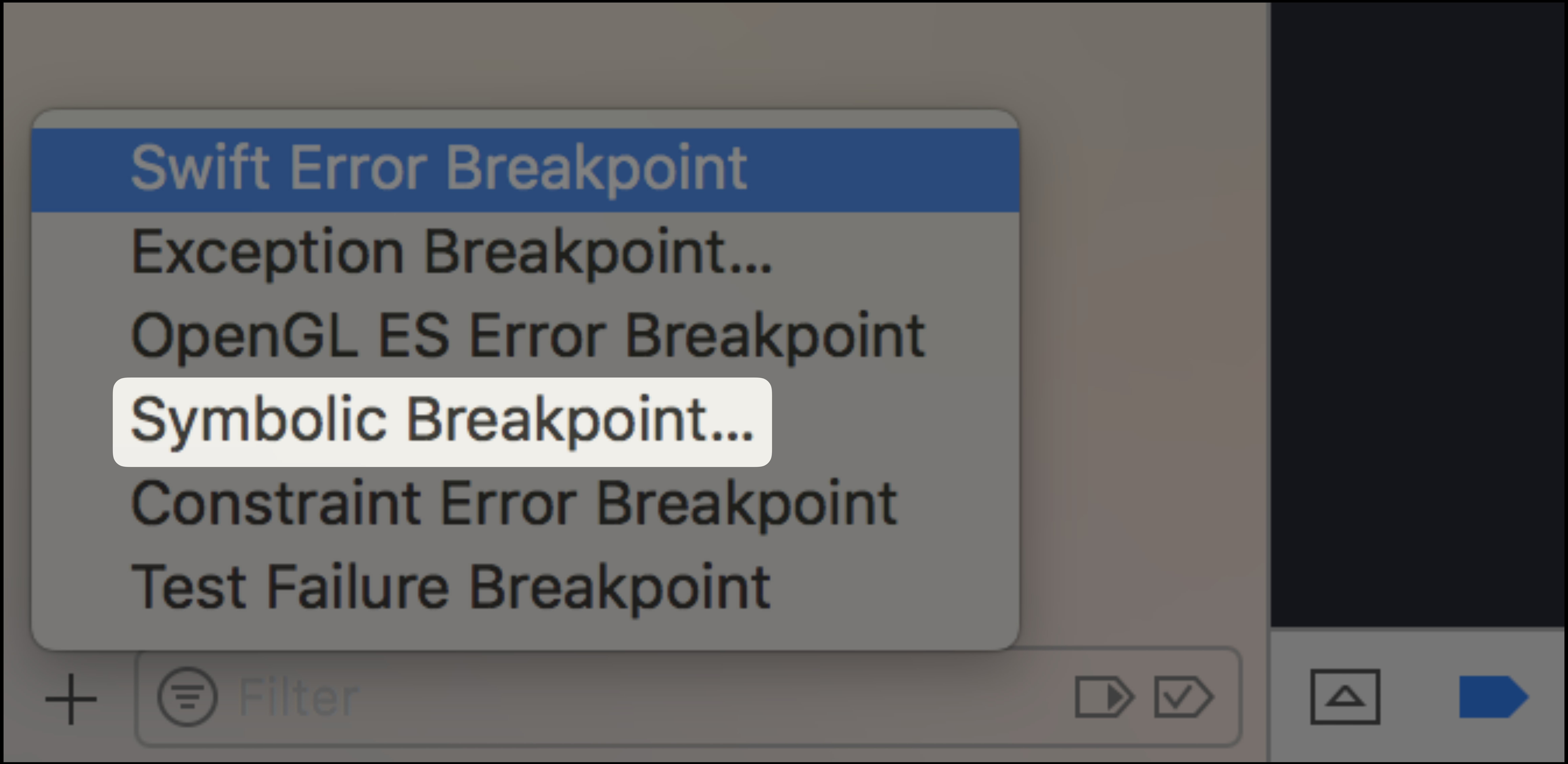
Use Case #5

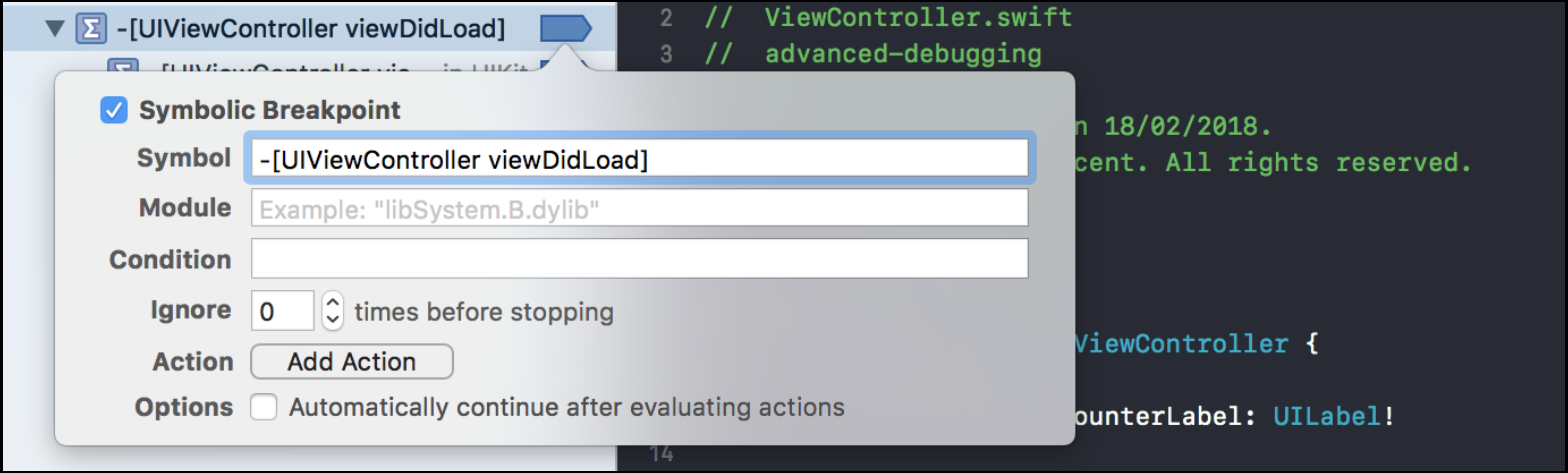
“I want to know when a framework method was called”

Might seem obvious: just put a breakpoint inside the method 🤓

Yeah, but you can’t do that for **framework** methods


Swift Error Breakpoint
Exception Breakpoint...
OpenGL ES Error Breakpoint
Symbolic Breakpoint...
Constraint Error Breakpoint
Test Failure Breakpoint
+
Filter


Swift Error Breakpoint
Exception Breakpoint...
OpenGL ES Error Breakpoint
Symbolic Breakpoint...
Constraint Error Breakpoint
Test Failure Breakpoint
+
Filter

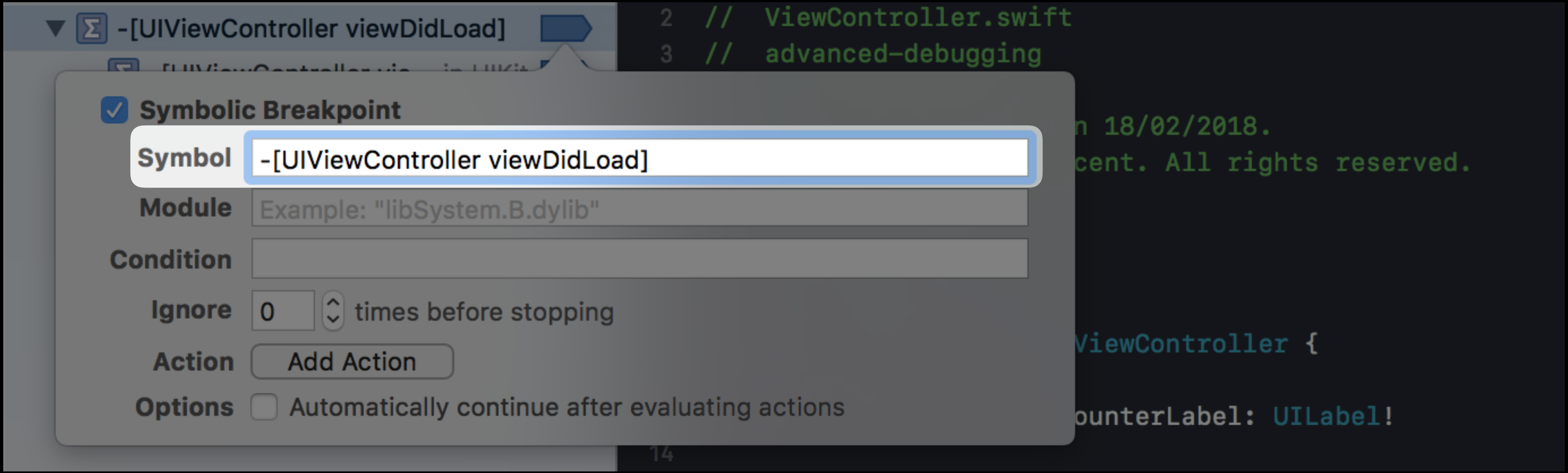


-[UIViewController viewDidLoad]

Symbolic Breakpoint [✓]

Symbol	-[UIViewController viewDidLoad]
Module	Example: "libSystem.B.dylib"
Condition	
Ignore	0 times before stopping
Action	Add Action
Options	Automatically continue after evaluating actions

```
//  ViewController.swift
//  advanced-debugging
n 18/02/2018.
cent. All rights reserved.
ViewController {
ounterLabel: UILabel!
```


-[UIViewController viewDidLoad]

Symbolic Breakpoint

Symbol: -[UIViewController viewDidLoad]

Module: Example: "libSystem.B.dylib"

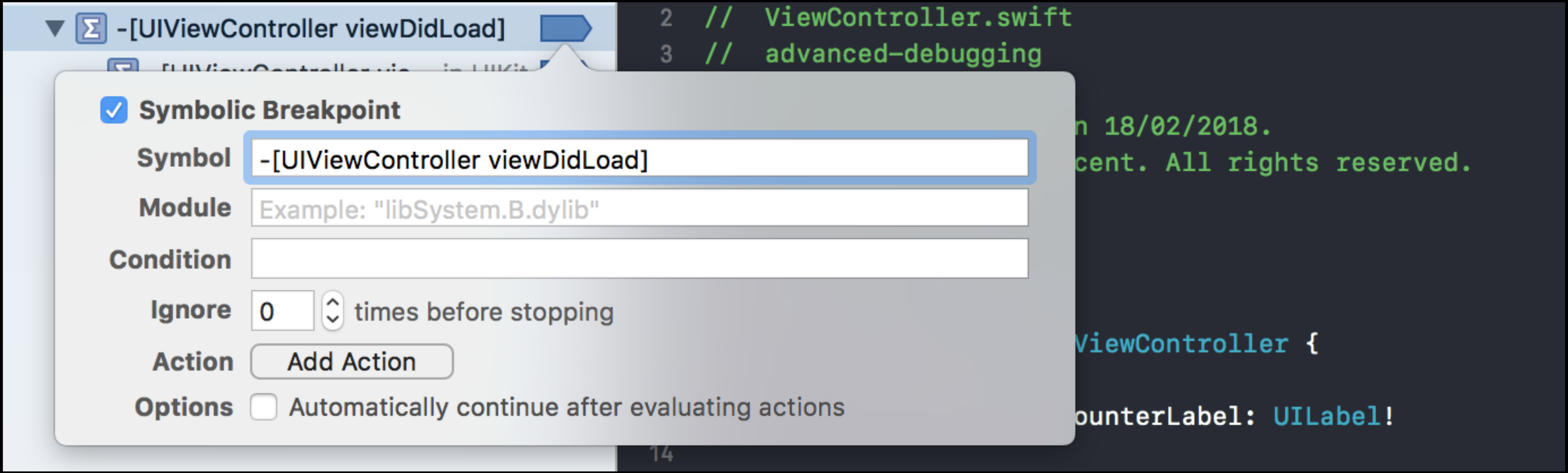
Condition:

Ignore 0 times before stopping

Action: Add Action

Options: Automatically continue after evaluating actions

```
//  ViewController.swift
//  advanced-debugging
n 18/02/2018.
cent. All rights reserved.
ViewController {
ounterLabel: UILabel!
```



-[UIViewController viewDidLoad]

Symbolic Breakpoint ✓

Symbol	-[UIViewController viewDidLoad]
Module	Example: "libSystem.B.dylib"
Condition	
Ignore	0 times before stopping
Action	Add Action
Options	☐ Automatically continue after evaluating actions

```
//  ViewController.swift
//  advanced-debugging
n 18/02/2018.
cent. All rights reserved.
ViewController {
ounterLabel: UILabel!
```

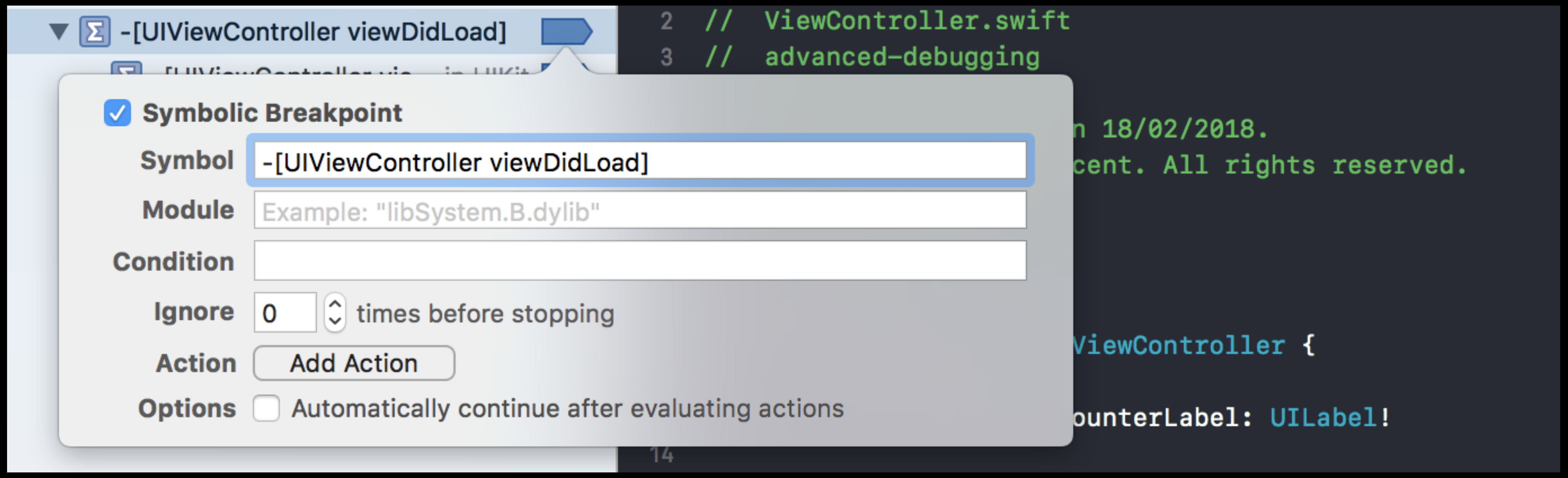
Symbolic breakpoints are crazy powerful! 🤯

Some controller gets presented out of nowhere?Just add a breakpoint to:

```
-[UIViewController presentViewController:animated:completion:]
```

Want to know if a controller is leaking? Breakpoint on this method:

```
-[UIViewController dealloc]
```



-[UIViewController viewDidLoad]

Symbolic Breakpoint

Symbol: -[UIViewController viewDidLoad]

Module: Example: "libSystem.B.dylib"

Condition:

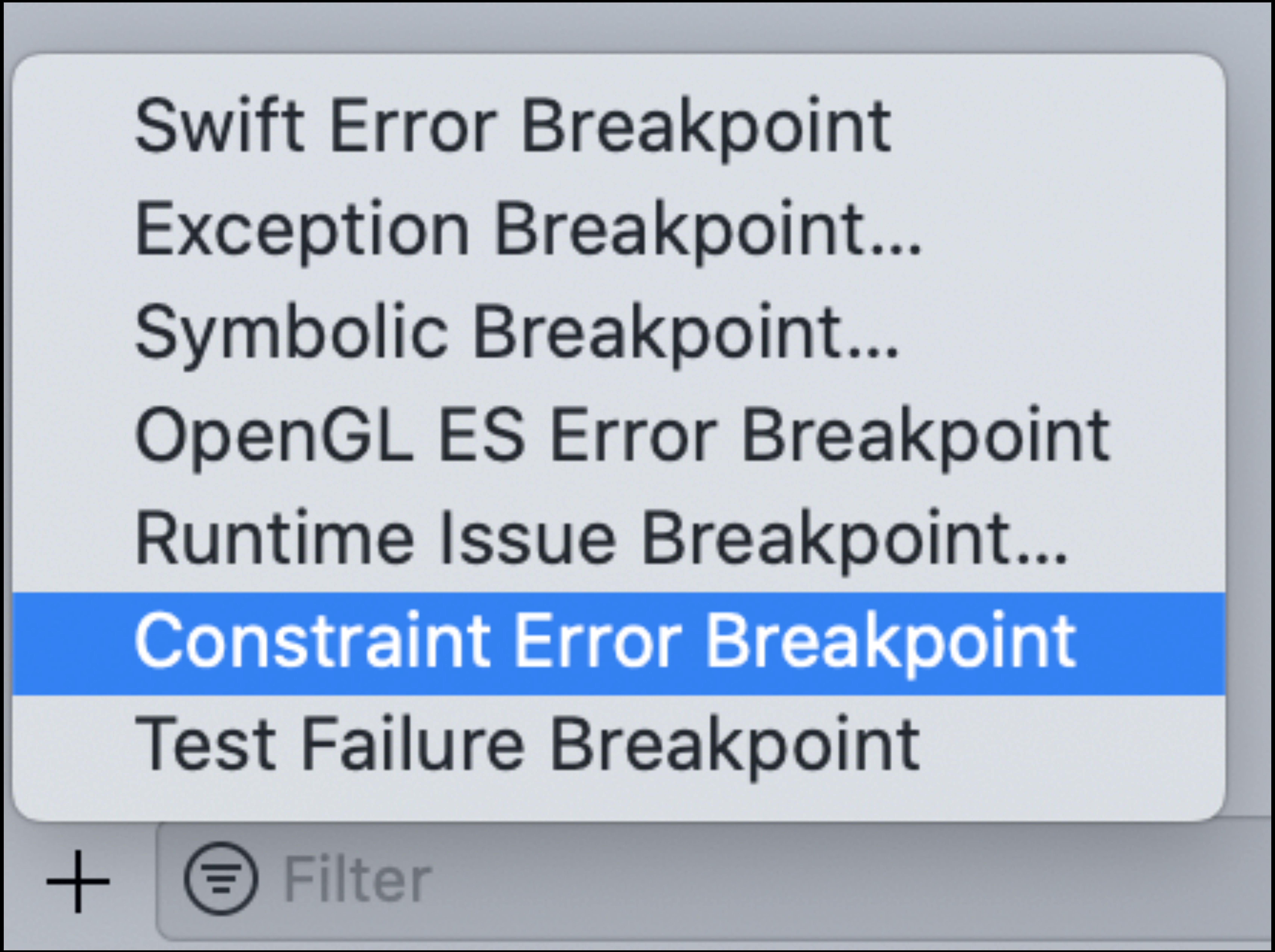
Ignore: 0 times before stopping

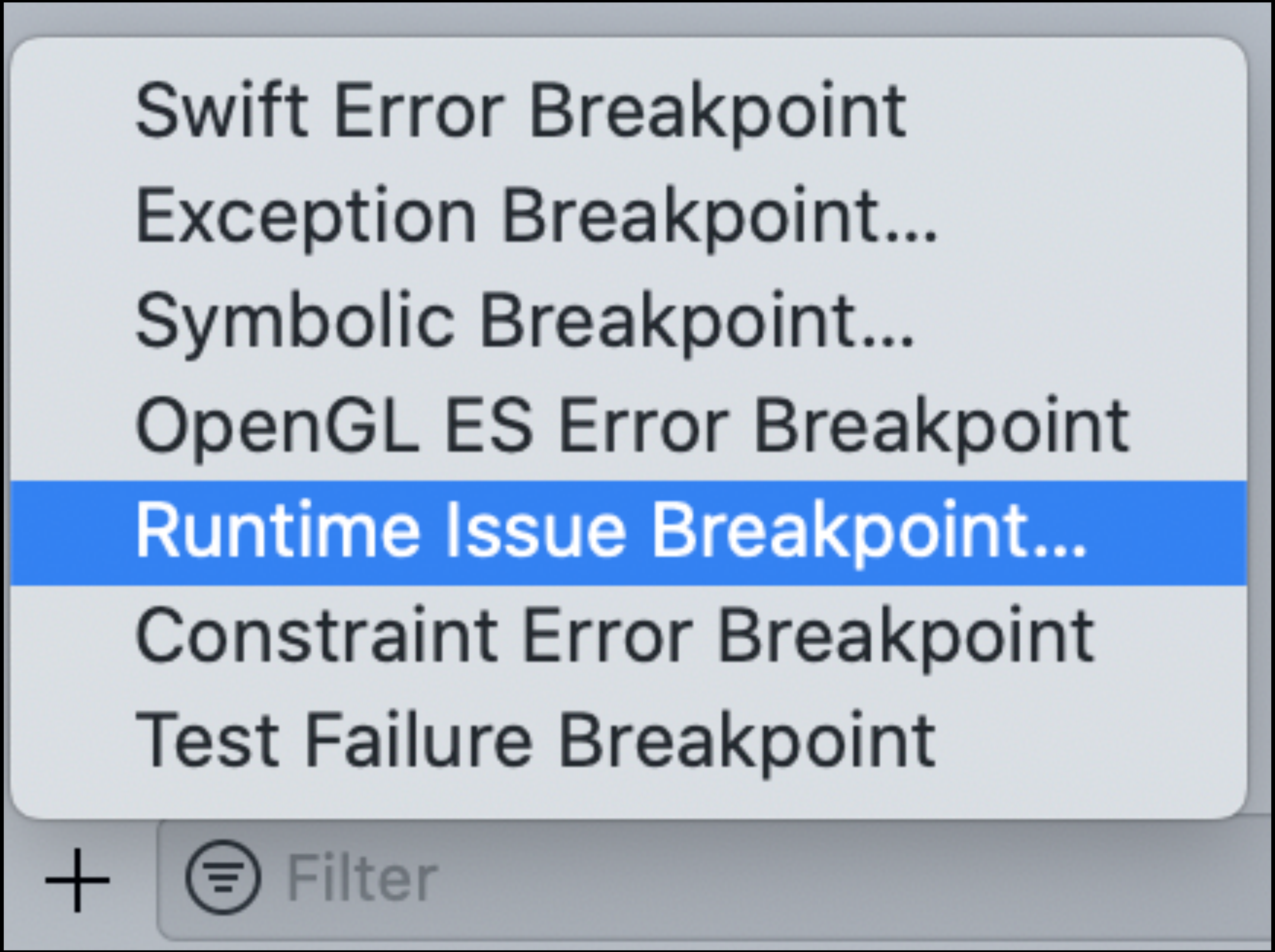
Action: Add Action

Options: Automatically continue after evaluating actions

```
//  ViewController.swift
//  advanced-debugging
```

There’s even more useful stuff to go through!


Swift Error Breakpoint
Exception Breakpoint...
Symbolic Breakpoint...
OpenGL ES Error Breakpoint
Runtime Issue Breakpoint...
Constraint Error Breakpoint
Test Failure Breakpoint
Filter


Swift Error Breakpoint
Exception Breakpoint...
Symbolic Breakpoint...
OpenGL ES Error Breakpoint
Runtime Issue Breakpoint...
Constraint Error Breakpoint
Test Failure Breakpoint
Filter

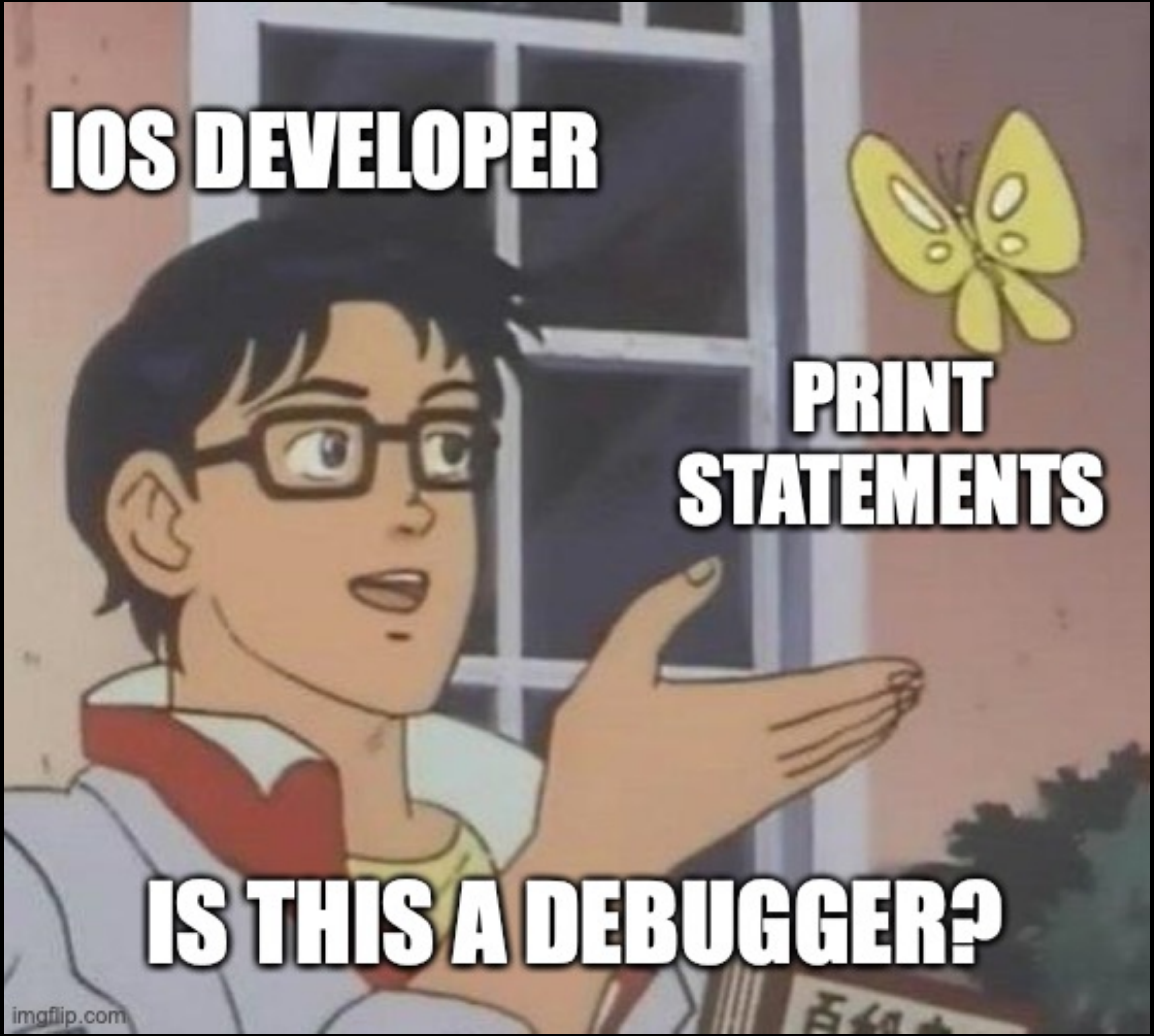
System Frameworks (Runtime Iss...

- [✓] Syste… …sue)
- Ignor… …pping
- Typ…

```
// ViewController.swift
// Test
…on
…ince
import UIKit
```

Ex
Thread Sanitizer
Undefined Behavior
Main Thread Checker
✓ System Frameworks
All

One More Thing!


IOS DEVELOPER
PRINT STATEMENTS
IS THIS A DEBUGGER?
imgflip.com

The article this talk is based on

👇

https://medium.com/flawless-app-stories/advanced-debugging-with-xcode-5e6c8dabd311

The Secret Life of Xcode Breakpoints

Vincent Pradeilles (@v_pradeilles) – Worldline