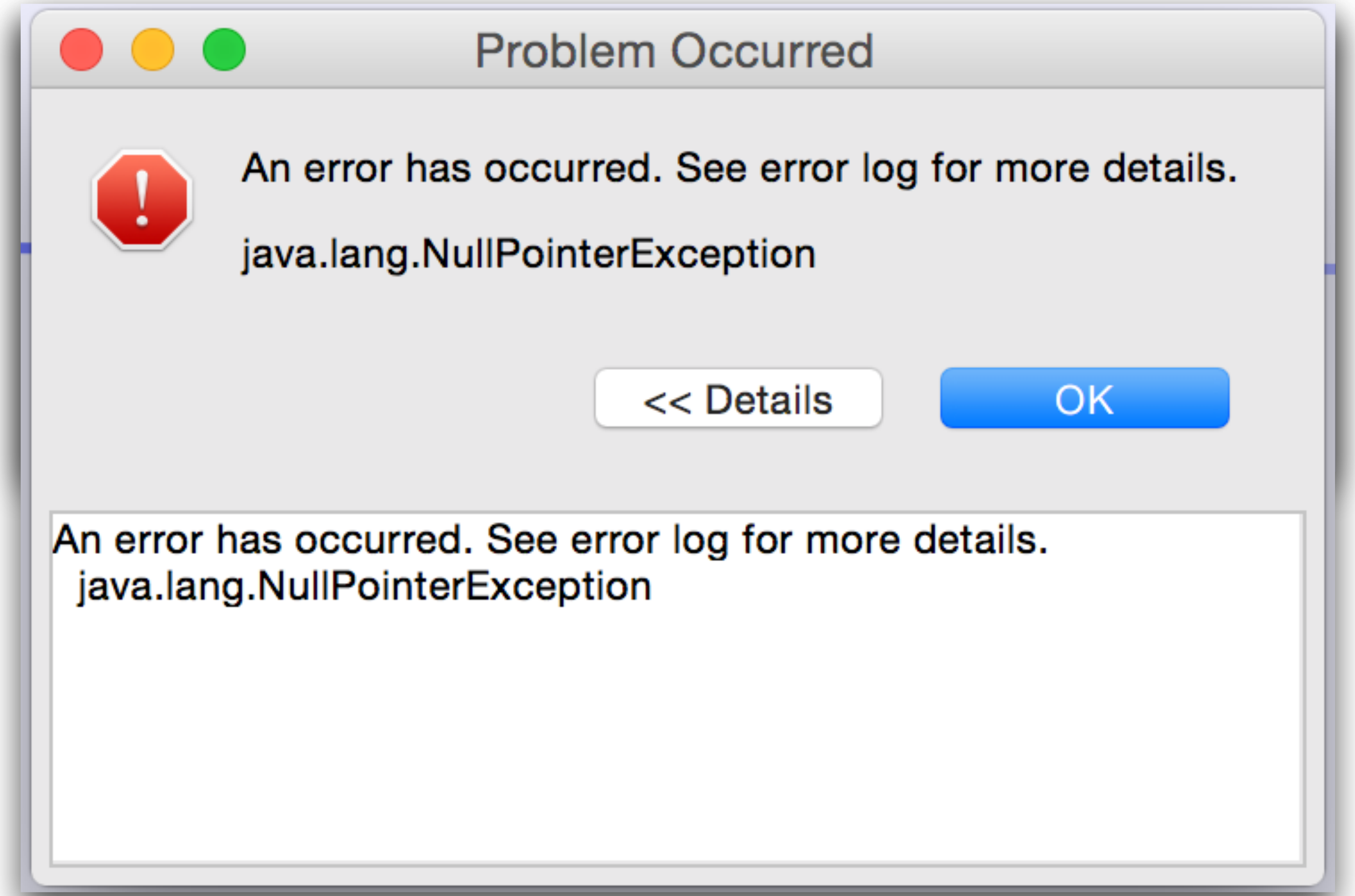
Nullability

Billion Dollar Mistake



An error has occurred. See error log for more details.

java.lang.NullPointerException

```
An error has occurred. See error log for more details.
 java.lang.NullPointerException
```

Modern approach:
to make NPE
compile-time error,
not run-time error

Nullable types in Kotlin

```
val s1: String = null       ✗

val s2: String? = "can be null or non-null"


s1.length    ✓

s2.length    ✗
```

Dealing with Nullable Types

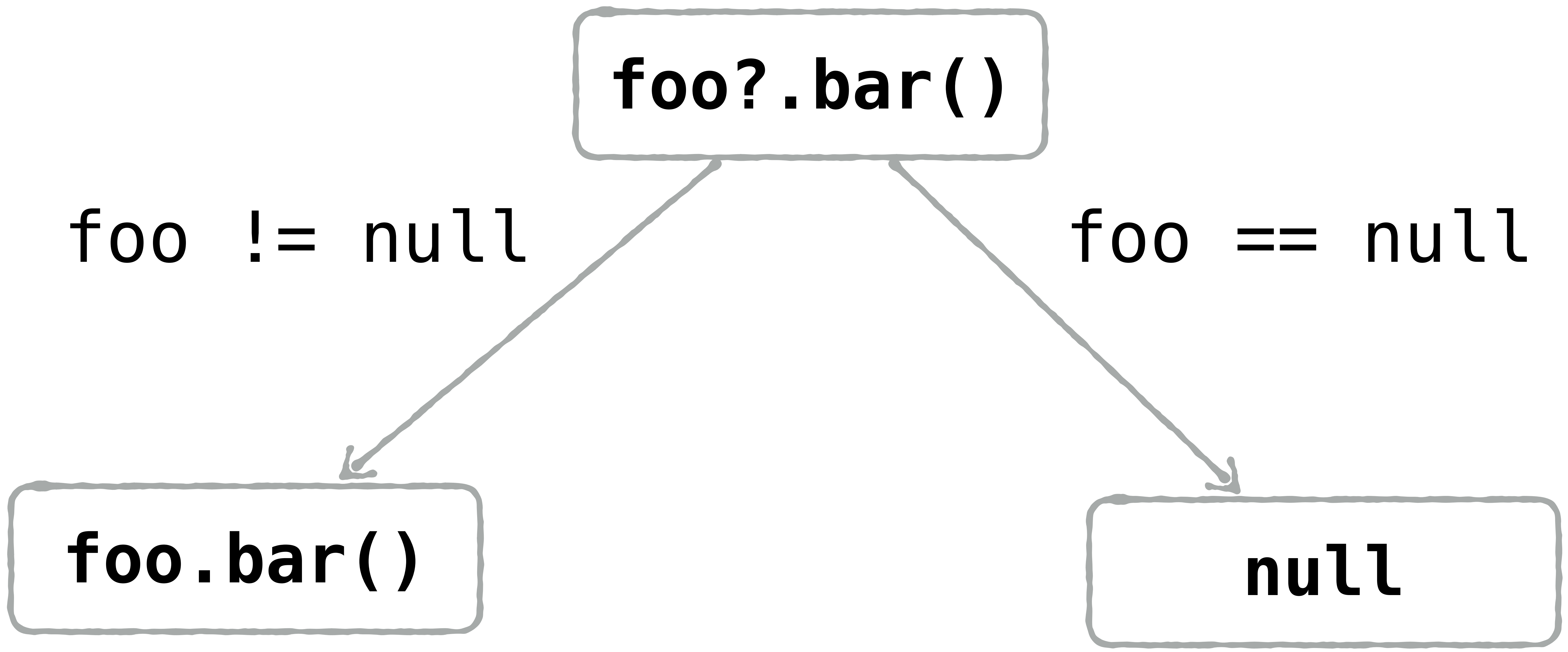
```
val s: String?

if (s != null) {
   s.
}
```

Replace 'if' expression with safe access expression

```
s?.length
```

Safe access



Nullability operators

```
val s: String?


val length = if (s != null) s.length else null
```

$\downarrow$

```
val length = s?.length
```

Which type does `length` variable below have?

```
val s: String?
...
val length = s?.length
```

?

Which type does `length` variable below have?

```
val s: String?
...
val length = s?.length
```

Int?

Nullability operators

```
val s: String?

val length: Int? = if (s != null) s.length else null
```

↓

```
val length: Int? = s?.length
```

Nullability operators

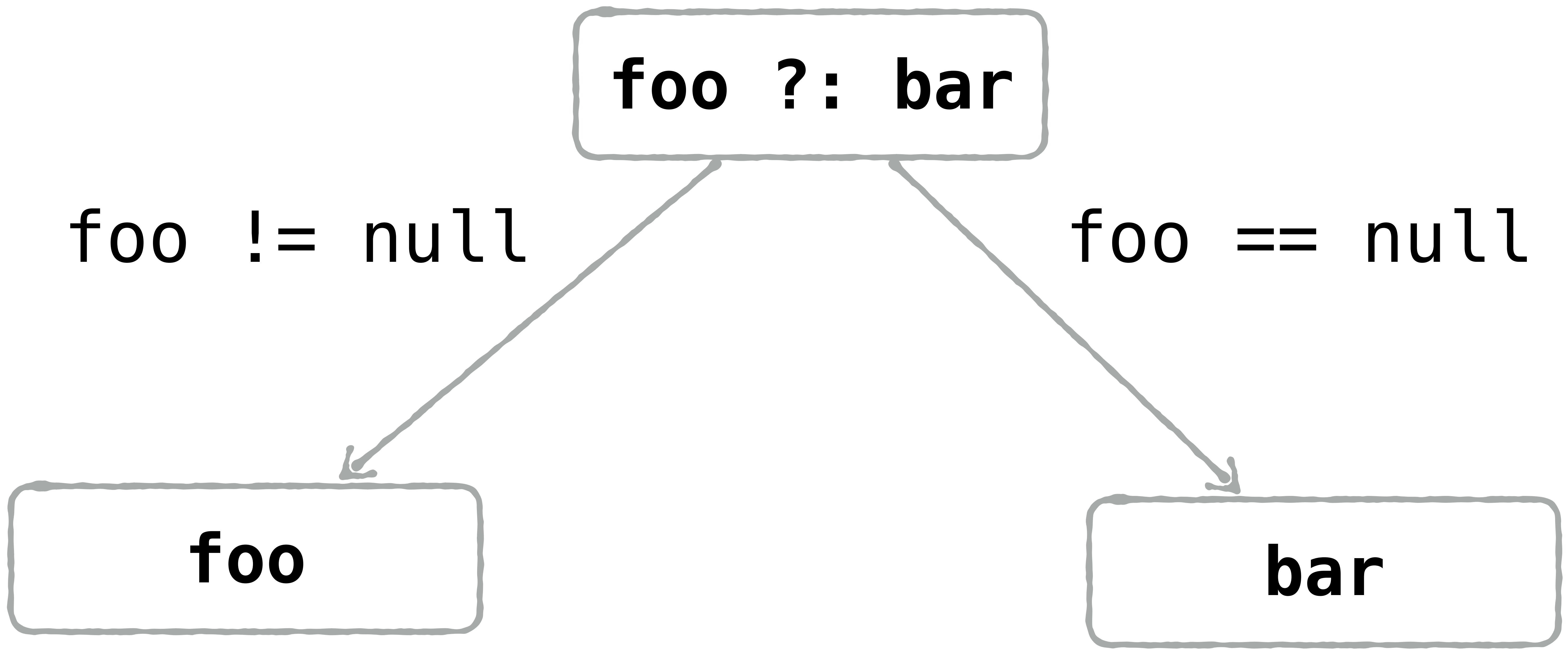
```kotlin
val s: String?

val length: Int = if (s != null) s.length else 0
```

↓

```kotlin
val length: Int = s?.length ?: 0
```

Elvis operator



Why “elvis operator”?

?:

:-)

What will be printed?

```kotlin
val a: Int? = null
val b: Int? = 1
val c: Int = 2

val s1 = (a ?: 0) + c
val s2 = (b ?: 0) + c
print("$s1$s2")
```

?

What will be printed?

```kotlin
val a: Int? = null
val b: Int? = 1
val c: Int = 2

val s1 = (a ?: 0) + c   // 2
val s2 = (b ?: 0) + c   // 3
print("$s1$s2")
```

$\boxed{23}$

Control-flow analysis

```
val s: String?


if (s == null) fail()
s.length
```

smart cast

Control-flow analysis

```
val s: String?


if (s == null) return
s.length
```

smart cast

Making NPE explicit

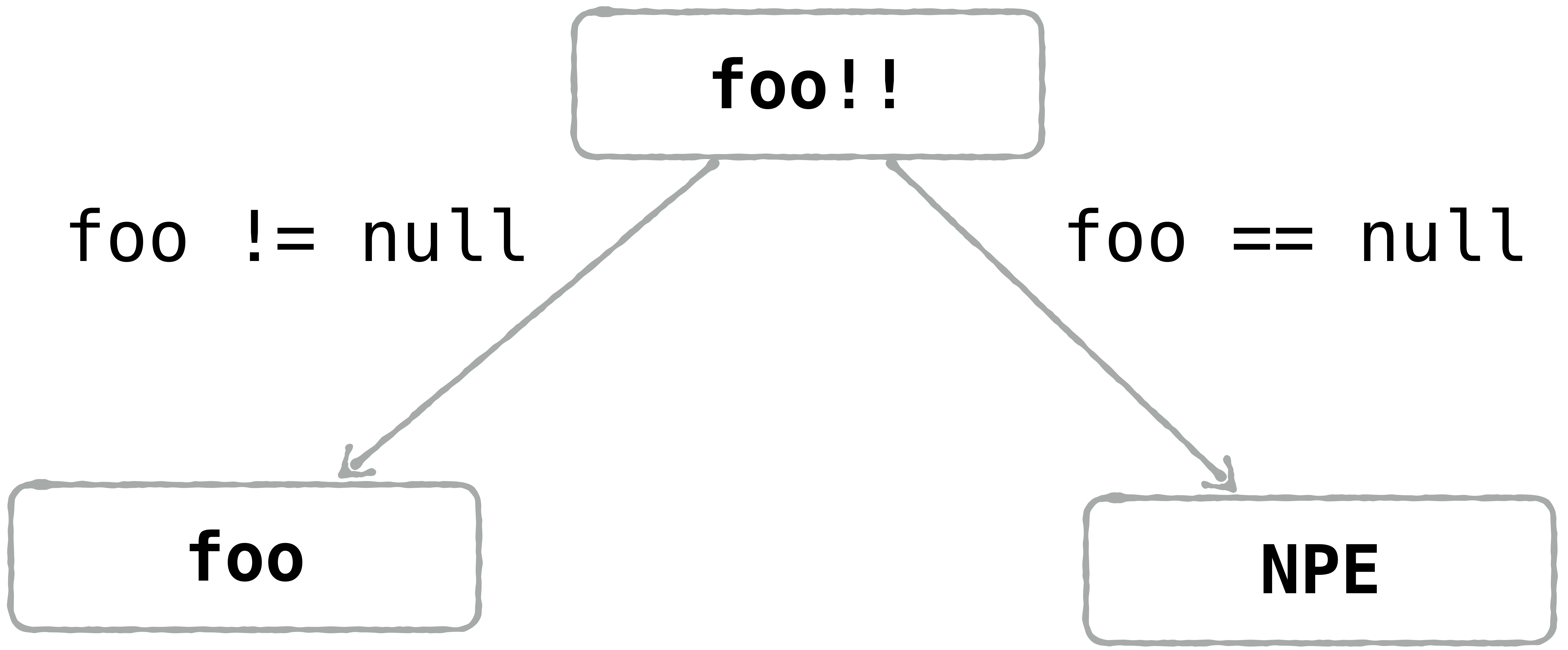
```
val s: String?
```

throws NPE if s is null

```
s!!
```

```
s!!.length
```

Not-null assertion



Which line(s) won’t compile?

```
#1 fun isFoo1(n: Name) = n.value == "foo"
#2 fun isFoo2(n: Name?) = n.value == "foo"
#3 fun isFoo3(n: Name?) = n != null && n.value == "foo"
#4 fun isFoo4(n: Name?) = n?.value == "foo"

   fun main(args: Array<String>) {
#5     isFoo1(null)
#6     isFoo2(null)
#7     isFoo3(null)
#8     isFoo4(null)
   }
```

?

Which line(s) won’t compile?

```
#1 fun isFoo1(n: Name) = n.value == "foo"

#5 isFoo1(null)

     Compiler error: Null can not be
     a value of a non-null type Name
```

Which line(s) won't compile?

```
#2 fun isFoo2(n: Name?) = n.value == "foo"

#6 isFoo2(null)
```

```
Compiler error: Only safe (?.) or
non-null asserted (!!.) calls are allowed
on a nullable receiver of type Name?
```

Which line(s) won't compile?

```
#3 fun isFoo3(n: Name?) = n != null && n.value == "foo"

#7 isFoo3(null)
```

✓

Which line(s) won't compile?

```
#4 fun isFoo4(n: Name?) = n?.value == "foo"

#8 isFoo4(null)
```

✓

Which line(s) won’t compile?

```
#1 fun isFoo1(n: Name) = n.value == "foo"
#2 fun isFoo2(n: Name?) = n.value == "foo"
#3 fun isFoo3(n: Name?) = n != null && n.value == "foo"
#4 fun isFoo4(n: Name?) = n?.value == "foo"

   fun main(args: Array<String>) {
#5     isFoo1(null)
#6     isFoo2(null)
#7     isFoo3(null)
#8     isFoo4(null)
   }
```

2, 5

Puzzler. What will be printed?

```kotlin
val x: Int? = 1
val y: Int = 2
val sum = x ?: 0 + y
println(sum)
```

1. 1
2. 2
3. 3

?

Puzzler. What will be printed?

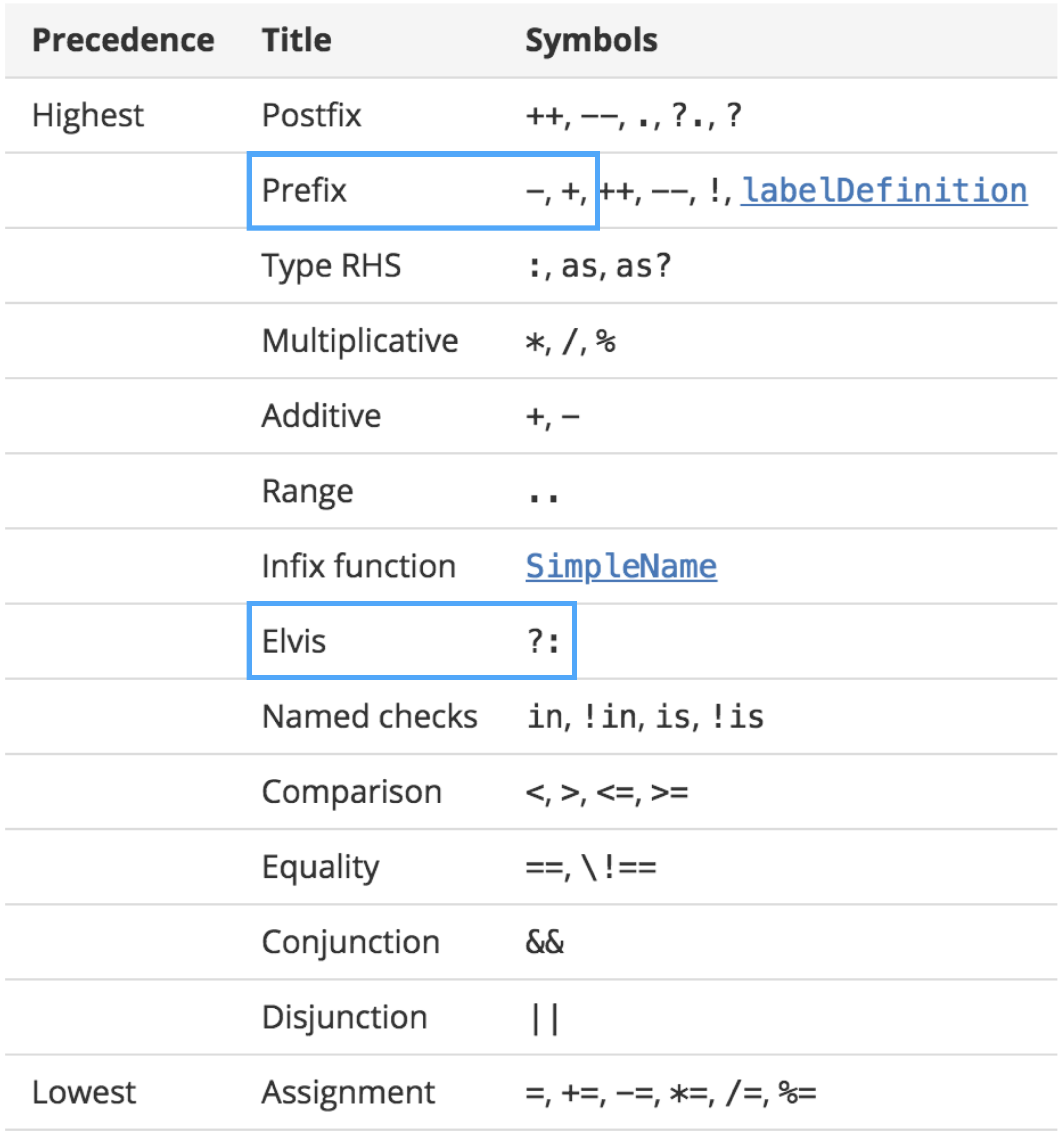
```kotlin
val x: Int? = 1
val y: Int = 2
val sum = x ?: 0 + y
println(sum)
```

[✓ 1. 1]
2. 2
3. 3

Operator precedence

```
val x: Int? = 1
val y: Int = 2
val s1 = x ?: 0 + y          // 1
val s2 = x ?: (0 + y)        // 1
```

Operator precedence



Precedence	Title	Symbols
Highest	Postfix	++, --, ., ?., ?
	Prefix	-, +, ++, --, !, labelDefinition
	Type RHS	:, as, as?
	Multiplicative	*, /, %
	Additive	+, -
	Range	..
	Infix function	SimpleName
	Elvis	?:
	Named checks	in, !in, is, !is
	Comparison	<, >, <=, >=
	Equality	==, \!==
	Conjunction	&&
	Disjunction	||
Lowest	Assignment	=, +=, -=, *=, /=, %=

Prefer parentheses

```
val x: Int? = 1

val y: Int = 2

val s1 = x ?: 0 + y       // 1

val s2 = x ?: (0 + y)     // 1

val s3 = (x ?: 0) + y   // 3
```