Nullability

Kevin Jones

@kevinrjones

Overview

The ‘billion dollar problem’

Kotlin’s approach to nullability

Operators that support nullability

Nullability

Java often throws NullPointerException

Need lots of defensive code

Kotlin supports 'nullable' types

Means only explicit variables can be null

Collections also support null constraints

```
boolean closeMeeting(Meeting meeting) {

    if (meeting.canClose) return meeting.close();

    return false;

}
```

Is this function safe?

What happens if null is passed?

Can null be passed?

```
fun closeMeeting(m: Meeting): Boolean {
    return if (m.canClose) m.close()
        else false
}
```

Is this function safe?

What happens if null is passed?

Can null be passed?

```
closeMeeting(null)
```

```
Error:(7, 18) Kotlin: Null can not be a value of a non-null type
Meeting
```

Try Calling with Null

Code does not compile

What if you want to allow null?

Variable can be declared to accept null

Put '?' after type name

val m: Meeting?

```
val m:Meeting? = null

m.close()
```

```
Error:(6, 6) Kotlin: Only safe (?.) or non-null asserted (!!.) calls
are allowed on a nullable receiver of type Meeting?
```

Try Calling with Null

Code does not compile

Safe Call (?.)

m?.method()

Checks for null

Calls function only if non-null reference

```
val m:Meeting? = null

val newMeeting = Meeting()

newMeeting = m;
```

```
Error:(7, 18) Kotlin: Type mismatch: inferred type is Meeting? but
Meeting was expected
```

Assigning Values

Cannot assign to non-null type

Null coalescing (?:)

Also known as the ‘Elvis’ operator

newMeeting = m?: Meeting()

Returns value on null

Safe cast (as?)

```
val saveable = o as? ISaveable
```

Casts to type or returns null

Not-Null assertions (!!)

m!!.close()

Very blunt instrument

Asserts that something is not null

Throws NPE if it is

Throws at line operator is used (not later)

It's meant to stand out!!

‘let’ function

Used to avoid explicit null checks

Useful when passing Nullable values ...

... to functions expecting non-null values

Late initialized properties

Sometimes values cannot be initialized when declared

Do not want to mark them as Nullable

Use 'lateinit' instead

Summary

Great support for non-null values in Kotlin

- Types have to be declared as being able to be null
- e.g. String? Vs String

Have null aware operators

- Safe call (?.)
- Safe cast (as?)
- Null coalescing (?:)
- Non-null assertions

Summary

Other support

- ‘let’ for safely passing null values
- lateinit to initialize properties in a class

Java Interoperability