Nullable types

&

Nullable Types Under the Hood

`@Nullable`, `@NotNull` annotations

Nullability annotations

`@Nullable` `Type` $\longrightarrow$ `Type?`

`@NotNull` `Type` $\longrightarrow$ `Type`

Nullability & Java

$$\begin{tikzcd} \boxed{\text{Type}} \arrow[r] & \boxed{?} \end{tikzcd}$$

behaves like
regular Java type

Platform type

notation, not syntax

`Type` $\longrightarrow$ **`Type!`**

type that came from Java

type of “unknown” nullability

A bit of history…

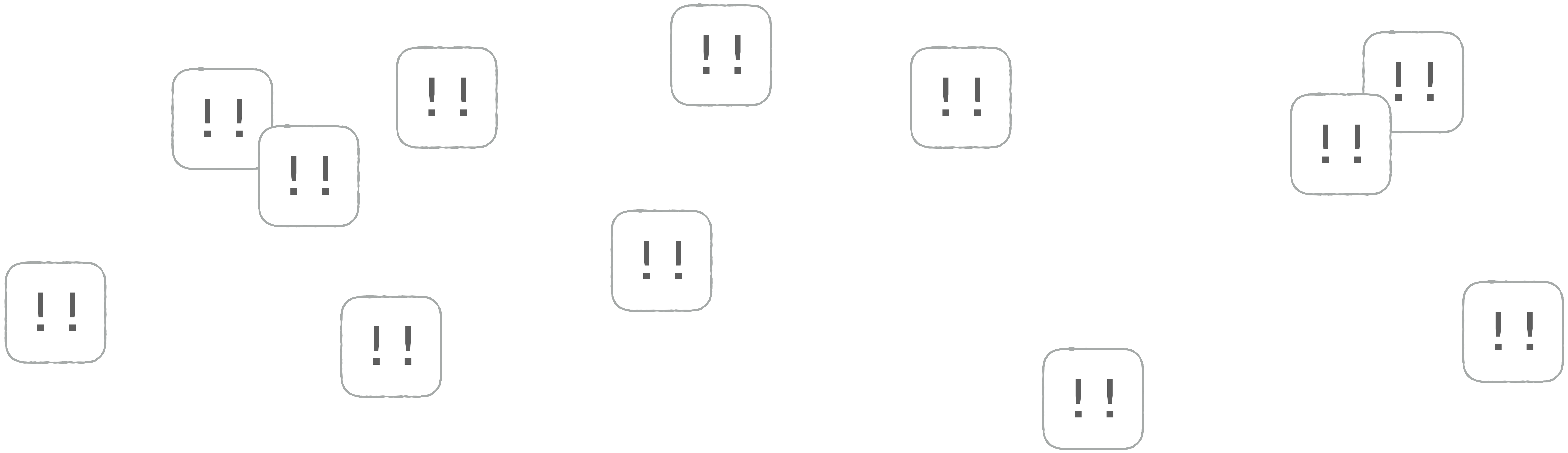
The safest approach would be…

$$\begin{tikzcd} \boxed{\texttt{Type}} \arrow[r] & \boxed{\texttt{Type?}} \end{tikzcd}$$

We tried, but it didn’t work well

If Type $\longrightarrow$ Type?

then the code looks like



And it doesn't really work with generics

Platform type

`Type` $\longrightarrow$ `Type!`

type that came from Java

type of “unknown” nullability

Platform type in error message

```
public class Session {
    public String getDescription()
}
```

```
val session = Session()
val description: Boolean = session.description
```

```
Compiler error: Type mismatch:
   inferred type is String!
   but Boolean was expected
```

What happens while running the code below?

```
public class Session {
  public String getDescription() {
    return null;
  }
}
```

```
val session = Session()
val description = session.description
println(description.length)
```

1. `NullPointerException` is thrown
2. `null` is printed
3. `compilation error`
4. `IllegalStateException` is thrown

?

What happens while running the code below?

```java
public class Session {
  public String getDescription() {
    return null;
  }
}
```

```kotlin
val session = Session()
val description = session.description
println(description.length)
```

1. [✓ `NullPointerException` is thrown]
2. `null` is printed
3. `compilation error`
4. `IllegalStateException` is thrown

Using Java from Kotlin

```java
public class Session {
  public String getDescription() {
    return null;
  }
}
```

```kotlin
: String!
val session = Session()
val description = session.description
println(description.length)
```

NullPointerException!

Using Java from Kotlin

```java
public class Session {
  public String getDescription() {
    return null;
  }
}
```

: String!

```kotlin
val session = Session()
val description = session.description
println(description?.length)
```

✓

How to still prevent NPEs?

- Annotate your Java types
- Specify types explicitly

How to still prevent NPEs?

- Annotate your Java types
- Specify types explicitly

Annotate your Java types

$$\begin{tikzcd}
\texttt{@Nullable Type} \arrow[r] & \texttt{Type?} \\
\texttt{@NotNull Type} \arrow[r] & \texttt{Type}
\end{tikzcd}$$

Different annotations are supported

`@Nullable`	`@NotNull`	JetBrains
`@Nullable`	`@NonNull`	Android
`@Nullable`	`@CheckForNull`	JSR-305
`@Nullable`	`@CheckForNull`	FindBugs
	`@NonNull`	Lombok

...

What happens while running the code below?

```java
public class Session {
    @Nullable
    String getDescription() {
        return null;
    }
}
```

```kotlin
val session = Session()
val description = session.description
println(description.length)
```

1. `NullPointerException` is thrown
2. `null` is printed
3. `compilation error`
4. `IllegalStateException` is thrown

?

What happens while running the code below?

```java
public class Session {
    @Nullable
    String getDescription() {
        return null;
    }
}
```

```kotlin
val session = Session()
val description = session.description
println(description.length)
```

1. `NullPointerException` is thrown
2. `null` is printed
3. [✓ `compilation error`]
4. `IllegalStateException` is thrown

Using Java from Kotlin

```java
public class Session {
    @Nullable
    String getDescription() {
        return null;
    }
}
```

: String?

```kotlin
val session = Session()
val description = session.description
println(description.length)
```

compiler error

Using Java from Kotlin

```java
public class Session {
    @Nullable
    String getDescription() {
        return null;
    }
}
```

: String?

```kotlin
val session = Session()
val description = session.description
println(description?.length)
```

✓

Annotate your Java types

`@Nullable` `Type`

`@NotNull` `Type`

- All of them???

- You can specify **`@NotNull`** as default, and annotate only **`@Nullable`** types

Non-null by default (JSR-305)

```
@ParametersAreNonnullByDefault
```

```
@MyNonnullByDefault
```

@MyNonnullApi

```
@javax.annotation.Nonnull
@TypeQualifierDefault(ElementType.PARAMETER, ...)
annotation class MyNonnullByDefault
```

package-info.java

```
@MyNonnullByDefault
package mypackage;
```

@NonNull by default

```
@MyNonnullByDefault
public class Session {
    public void setDescription(String description) {
        this.description = description;
    }
}
```

```
val session = Session()
session.setDescription(null)
```

```
Warning: Expected type doesn’t accept nulls in Java,
         but the value may be null in Kotlin
```

Make it an error

build.gradle

```
compileKotlin {
   kotlinOptions {
        freeCompilerArgs += "-Xjsr305=strict"
    }
}
```

@NonNull by default

```
@MyNonnullByDefault
public class Session {
    public void setDescription(String description) {
        this.description = description;
    }
}
```

```
val session = Session()
session.setDescription(null)
```

```
Error: Null can not be a value of a non-null type String
```

How to still prevent NPEs?

- Annotate your Java types
- Specify types explicitly

What happens while running the code below?

```java
public class Session {
  public String getDescription() {
    return null;
  }
}
```

```kotlin
val session = Session()
val description: String? = session.description
println(description?.length)
```

1. `NullPointerException` is thrown
2. `null` is printed
3. `compilation error`
4. `IllegalStateException` is thrown

?

What happens while running the code below?

```java
public class Session {
  public String getDescription() {
    return null;
  }
}
```

```kotlin
val session = Session()
val description: String? = session.description
println(description?.length)
```

1. `NullPointerException` is thrown
2. [✓ `null` is printed]
3. `compilation error`
4. `IllegalStateException` is thrown

Specify types explicitly

```java
public class Session {
    public String getDescription() {
        return null;
    }
}
```

```kotlin
val session = Session()
val description: String? = session.description
println(description?.length)
```

✓

What happens while running the code below?

```java
public class Session {
  public String getDescription() {
    return null;
  }
}
```

```kotlin
val session = Session()
val description: String = session.description
println(description.length)
```

1. `NullPointerException` is thrown
2. `null` is printed
3. `compilation error`
4. `IllegalStateException` is thrown

?

What happens while running the code below?

```java
public class Session {
  public String getDescription() {
    return null;
  }
}
```

```kotlin
val session = Session()
val description: String = session.description
println(description.length)
```

1. `NullPointerException` is thrown
2. `null` is printed
3. `compilation error`
4. [✓ `IllegalStateException` is thrown]

Specify types explicitly

```java
public class Session {
  public String getDescription() {
    return null;
  }
}
```

```kotlin
val session = Session()
```

```kotlin
val description: String = session.description
                            IllegalStateException:
                            session.description must not be null
```

Intrinsic checks

```kotlin
val session = Session()
val description: String = session.description
println(description)
```

Intrinsic checks

```
val session = Session()
val description: String = session.description
```

```
Intrinsics.checkExpressionValueIsNotNull(
    description, "session.description");
```

```
println(description)
```

is generated by the compiler,
throws an exception if `session.description` is `null`

Intrinsic checks

```
public fun foo(s: String) {
    Intrinsics.checkParameterIsNotNull(s, "s");

}
```

How to still prevent NPEs?

- Annotate your Java types
- Specify types explicitly

Nullable platform types: summary

Good compromise between
safety and convenience