Kotlin Nullability

Here's an in-depth set of notes covering Kotlin nullability, combining all the examples from the slides with detailed explanations:

Overview

Kotlin introduces robust null safety features to avoid **null pointer exceptions (NPEs)**, one of the most common runtime errors in programming. By making nullability explicit in the type system, Kotlin ensures that developers handle nullable values properly during compile time.

---

1. Null's Use

Variables in Kotlin can either hold a **non-nullable** or a **nullable** type. By default, variables cannot hold `null`.

Example:

```
const val favoriteActor = "Sandra Oh"

fun main() {
    var favoriteActor = null  // This is allowed due to type inference in
scope.
    println(favoriteActor)    // Output: null
}
```

Here, `null` is assigned to a variable that does not have a defined type, and Kotlin allows this temporarily.

---

2. Non-Nullable Types

By default, Kotlin does not allow assigning `null` to variables with non-nullable types.

Example:

```
fun main() {
    var favoriteActor: String = "Sandra Oh"
    favoriteActor = null  // Compile-time error!
}
```

In this case, the type `String` is non-nullable, meaning it cannot hold a `null` value.

---

3. Nullable Types

To allow a variable to hold `null`, you must explicitly declare it as nullable by appending `?` to the type.

Example:

```
fun main() {
    var favoriteActor: String? = "Sandra Oh"
    favoriteActor = null  // No error!
    println(favoriteActor)  // Output: null
}
```

When a type is nullable (`String?`), it can safely hold `null`.

---

4. Null Safety

Kotlin prioritizes compile-time errors over runtime errors by enforcing null safety during code compilation.

Problematic Code Example:

```
fun main() {
    var favoriteActor: String? = "Sandra Oh"
    println(favoriteActor.length)  // Compile-time error!
}
```

Even though `favoriteActor` is initialized, its type is `String?`, so accessing its properties directly without null checking is not allowed.

---

5. Safe Call Operator (`?.`)

The safe call operator is used to access a nullable object's properties or functions **only if it is not null**.

Example:

```kotlin
fun main() {
    var favoriteActor: String? = "Sandra Oh"
    println(favoriteActor?.length)  // Output: 9

    favoriteActor = null
    println(favoriteActor?.length)  // Output: null
}
```

- When the object is `null`, the safe call operator returns `null` instead of throwing an exception.

---

6. Not-Null Assertion (`!!`)

The `!!` operator asserts that a nullable variable is not `null`. However, if the variable is `null`, it throws a **NullPointerException** at runtime.

Example:

```kotlin
fun main() {
    var favoriteActor: String? = "Sandra Oh"
    println(favoriteActor!!.length)  // Output: 9

    favoriteActor = null
    println(favoriteActor!!.length)  // Throws NullPointerException!
}
```

- **Use `!!` only when you are absolutely certain the variable is not `null`.**

---

7. Checking for Null

Manually checking for null allows you to perform actions conditionally, based on whether a value is `null`.

Example:

```kotlin
fun main() {
    var favoriteActor: String? = null

    if (favoriteActor != null) {
        println("Number of characters in the name:
${favoriteActor.length}.")
    } else {
        println("You didn't input a name.")  // Output: You didn't input a
name.
    }
}
```

---

8. Typical Kotlin Null-Checking Example

Combining null checking with variable assignments:

Example:

```
fun main() {
    var favoriteActor: String? = "Sandra Oh"

    val lengthOfName = if (favoriteActor != null) {
        favoriteActor.length
    } else {
        0
    }

    println("Number of characters: $lengthOfName")  // Output: 9
}
```

---

9. Elvis Operator (`?:`)

The Elvis operator is a concise way to handle nullable values. It provides a default value to return when a nullable value is `null`.

Example:

```
fun main() {
    var favoriteActor: String? = "Sandra Oh"

    val lengthOfName = favoriteActor?.length ?: 0
    println("Number of characters: $lengthOfName")  // Output: 9

    favoriteActor = null
    println("Number of characters: ${favoriteActor?.length ?: 0}")  //
```

```
  Output: 0
}
```

- If `favoriteActor` is not `null`, its `length` is returned.
- If it is `null`, the default value (`0`) is returned.

Key Takeaways

1. **Non-nullable types**: By default, variables in Kotlin cannot hold `null` values, preventing null pointer exceptions.
2. **Nullable types**: Variables declared with `?` can safely hold `null` values.
3. **Safe call operator (`?.`)**: Prevents exceptions by returning `null` when accessing properties of a nullable object.
4. **Not-null assertion (`!!`)**: Use with caution to assert that a nullable variable is not `null`.
5. **Elvis operator (`?:`)**: Provides a concise way to handle nulls by specifying a default value.
6. **Prefer null safety at compile-time**: Kotlin enforces null safety at compile-time, reducing the risk of runtime errors.