TutorialKart

Kotlin – Null Safety in Kotlin

What is Null Safety in Kotlin?

Null Safety in Kotlin is to eliminate the risk of occurrence of NullPointerException in real time. If you are a Java developer, you might have definitely come across the atrocities of NullPointerException it creates during real time. NullPointerException is so popular as NPE (**N**ull**P**ointer**E**xception), since it costed billions of dollars to companies. Kotlin is so concerned about this, and made its type system in such a way that it eliminates the usage of NullPointerException, unless you want it explicitly. Thank you Kotlin.

Kotlin could never throw a NullPointerException unless you ask for it.

Of course there are ways you might give NullPointerException a chance to live one more Exception again. Let us see how could this happen :

- When you ask for a NullPointerException explicitly
  - to **throw a NullPointerException** or
  - use**!!** operator. This operator comes with a disclaimer for usage : You have been warned! And yet you chose to live with NullPointerException.
- From Outside Kotlin
  - You may be aware that you can run external Java code in your application. And this external Java Code is not NullPointerException proof, unless you make it so.
- Data inconsistency

How Kotlin handles Null Safety ?

Following are some of the ways to handle Null Safety in Kotlin.

- **Differentiate** between **nullable** references and **non-nullable** references.
- User explicitly **checks** for a **null** in conditions
- Using a **Safe Call** Operator (?.)
- Elvis Operator (?:) : If reference to a val is not null, use the value, else use some default value that is not null

Differentiate between Nullable and Non-nullable References

Kotlin's type system can differentiate between nullable references and non-nullable references. **?** operator is used during variable declaration for the differentiation.

To make a variable hold a null value, follow the below example and observe the usage of **?** in the program

Kotlin Program – example.kt

```
fun main(args: Array){
    var a: String = "Hello !"     // variable is declared as non-null by default
    println("a is : $a")
    // kotlin prevents you assign a null to a non-nullable variable
    // a=null //assing null to a causes complilation error

    var b: String? = "Hi !"     // variable is declared as nullable
    println("b is : $b")
    b = null
    println("b is : $b")
}
```

Output

```
Compilation completed successfully
a is : Hello !
b is : Hi !
b is : null
```

Explicitly Check for a Null in Condition

If condition is an expression in Kotlin and hence can return a value. We may use this to explicitly check for a null and return a value in a single statement.

In the below example, we shall check if the variable 'str' is null and access the properties of str if not null.

Kotlin Program – example.kt

```
fun main(args: Array){
    var b: String? = "Hello"     // variable is declared as nullable
    var blen = if(b!=null) b.length else -1
    println("b is : $b")
    println("b length is : $blen")

    b = null
    println("b is : $b")
    blen = if(b!=null) b.length else -1
    println("b length is : $blen")
}
```

Output

```
Compilation completed successfully
b is : Hi !
b length is : 4
b is : null
b length is : -1
```

Safe Call

The safe call operator returns the variables property only if the variable is not null, else it returns null. So the variable holding the return value should be declared nullable. Following is the example program where a safe call is made with the variable **b** on property **length**.

Kotlin Program – example.kt

```
fun main(args: Array){
    var b: String? = "Hi !"     // variable is declared as nullable
    var len: Int?
    len = b?.length
    println("b is : $b")
    println("length is : $len")

    b = null
    len = b?.length
    println("b is : $b")
    println("length is : $len")
}
```

Output

```
Compilation completed successfully
b is : Hi !
length is : 4
b is : null
length is : null
```

Elvis Operator (?:)

If reference to a variable is not null, use the value, else use some default value that is not null. This might sound same as explicitly checking for a null value. But the syntax could be reduced and is given below.

Kotlin Program – example.kt

```
fun main(args: Array){
    var b: String? = "Hello"     // variable is declared as nullable
    var blen = b?.length ?: -1
    println("b is : $b")
    println("b length is : $blen")

    b = null
    println("b is : $b")
    blen = b?.length ?: -1
    println("b length is : $blen")
}
```

Output

```
Compilation completed successfully
b is : Hello
b length is : 5
b is : null
b length is : -1
```

How to get a NullPointerException explicitly ?

Despite the safety measures Kotlin provides to handle NPE, if you need NPE so badly to include in the code, you have it. Following are the ways

- Throw an NPE if reference is null using (**!!**) operator
- During type casting (safely)

Throw an NPE if reference is null using (!!) operator

Consider the following example to throw a NullPointerException using !! operator.

Kotlin Program – example.kt

```
fun main(args: Array){
    var b: String? = "Hello"     // variable is declared as nullable
    var blen = b!!.length
    println("b is : $b")
    println("b length is : $blen")

    b = null
    println("b is : $b")
    blen = b!!.length
    println("b length is : $blen")
}
```

Output

```
Compilation completed successfully
b is : Hello
b length is : 5
b is : null

Exception in thread "main" kotlin.KotlinNullPointerException
    at ArrayaddremoveKt.main(Arrayaddremove.kt:9)
```

Conclusion

In this Kotlin Tutorial – Kotlin Null Safety, we have learnt what Null Safety is, differences between nullable references and non-nullable references, user explicitly checks for a null in conditions, using a Safe Call

references and non-nullable references, user explicitly checks for a null in conditions, using a Safe Call Operator (?.) and Elvis Operator (?:).

Kotlin Java

- Kotlin Tutorial

Getting Started

- Setup Kotlin(Java) Project
- Kotlin Example Program
- Convert Java to Kotlin
- Kotlin Main Function
- Kotlin Loops
- Kotlin For Loop
- Kotlin While, Do While Loops
- Kotlin Repeat
- Kotlin Ranges
- Kotlin When

Object Oriented Concepts

Classes

- Kotlin - Class, Primary and Secondary Constructors
- Kotlin Sealed Class
- Kotlin Data Class
- Kotlin Enum
- Kotlin - Extension Functions

Inheritance

- Kotlin Inheritance
- Kotlin Override Method of Super Class

Abstraction

- Kotlin Abstraction
- Kotlin Abstract Class
- Kotlin - Interfaces

- Kotlin - Interfaces

➪ **Kotlin Null Safety**

Exception Handling

- Kotlin Try Catch
- Kotlin Throw Exception
- Kotlin Custom Exception

Fix Compilation Errors

- Kotlin - Variable must be initialized
- Kotlin - Primary Constructor call expected
- Kotlin - Null can not be a value of a non-null type String
- Kotlin - Cannot create an instance of an abstract class

Kotlin - String Operations

- Kotlin - Compare Strings
- Kotlin - Replace String
- Kotlin - Split String
- Kotlin - Split String to Lines
- Kotlin - String Capitalize

Kotlin - Functions

- Kotlin Function - Default Arguments
- Kotlin - Use Function

Kotlin Collections

Kotlin List

- Kotlin List
- Kotlin List forEach

Kotlin File Operations

- Kotlin - Create File
- Kotlin - Read File
- Kotlin - Read File as List of Lines
- Kotlin - Write to File

- Kotlin - Write to File
- Kotlin - Append Text to File
- Kotlin - Check if File Exists
- Kotlin - Copy a File to Other
- Kotlin - Iterate through all files in a directory
- Kotlin - Delete Recursively
- Kotlin - Get File Extension

Kotlin Interview Q/A

- Kotlin Interview Questions

Kotlin Android

- Kotlin Android Tutorial

Useful Resources

- How to Learn Programming