**Null Safety in Kotlin**

**Nullable Types:**

A variable of a nullable type in Kotlin can hold a reference to an object of the specified type or a special value called null, which represents the absence of a value.

To declare a nullable type, you simply add a ? after the type name. For example, String? represents a nullable string.

**Example**:

val nullableString: String? = null

val length: Int? = nullableString?.length

**Non-Nullable Types:**

A non-nullable type in Kotlin is one that cannot hold a null value. This is the default behavior for most variable declarations. If you declare a variable without the ?, it's non-nullable.

When you have a non-nullable variable, the compiler enforces that you always initialize it with a non-null value. You can't assign null to a non-nullable variable.

**Example:**

val nonNullableString: String = "Hello, World!" // Valid

val nonNullableString2: String = null // Compile-time error

**Kotlin Null Safety Operator**

1. **Safe Call Operator (?.):**

The safe call operator allows you to safely access properties and call methods on nullable objects. If the object is null, the expression evaluates to null instead of throwing a null pointer exception.

val length: Int? = nullableString?.length

1. **Elvis Operator (?:):**

The Elvis operator provides a default value if an expression is null. It's used in conjunction with the safe call operator.

val length: Int = nullableString?.length ?: 0

1. **Not-Null Assertion Operator (!!):**

The not-null assertion operator is used to assert that a value is not null. Be cautious when using this operator because it can lead to a NullPointerException if the value is null.

val length: Int = nullableString!!.length

1. **Safe Cast Operator (as?):**

The safe cast operator is used to safely cast an object to another type. If the cast fails, it returns null instead of throwing a ClassCastException.

val number: Int? = someValue as? Int

1. **Null Coalescing Operator (?:):**

The null coalescing operator provides a way to choose a non-null value from a list of expressions. It returns the first non-null expression found.

val result = expression1 ?: expression2 ?: expression3

1. **Let Function:**

The let function allows you to perform an operation on a nullable object if it's not null. It takes a lambda as an argument and passes the object to the lambda.

nullableObject?.let { /\* perform an operation on it \*/ }

1. **Safe Access to Index in Arrays and Lists:**

When accessing elements at a specific index in an array or a list, you can use the safe indexing operator getOrNull. It returns null if the index is out of bounds.

val element = list.getOrNull(index)