



02



## 02-7. *Null Safety*



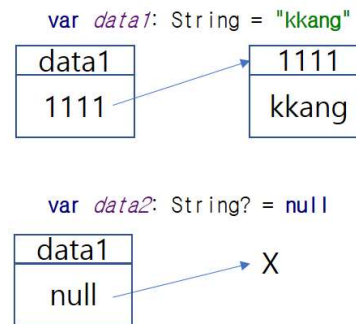
**Basic Syntax**

# Null 안전성

---

## Null 안전성이란?

- Null 이란 프로그램에서 값이 아무것도 대입되지 않은 상태



- The Billion Dollar Mistake
- Null-Safety란 Null에 다양한 처리를 도와줌으로서 Null에 의한 NPE이 발생하지 않는 프로그램의 작성을 작성할수 있게 해준다는 개념

# Null Safety - 타입

---

## null이 될 수 있는 변수와 null

- 코틀린에서는 null을 대입할 수 없는 변수와 있는 변수로 구분
- 변수에 null 값을 대입하려면 타입에 ? 기호를 이용하여 명시적으로 null이 될 수 있는 변수로 선언.

```
var data1: String = "kkang"

var data2: String? = null

fun main(args: Array<String>) {
    data1=null//error
}
```

```
var data1: String = "kkang"

var data2: String? = null

fun myFun(arg: String){
}

fun main(args: Array<String>) {
    data2="hello"

    val data3: String? = data1
    val data4: String=data2//error

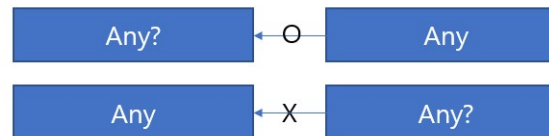
    myFun(data2)//error
}
```

# Null Safety - 타입

---

## Any, Any? 타입

- Any 타입으로 선언한 변수에는 null을 대입할 수 없지만, Any? 타입으로 선언한 변수에는 null을 대입할 수 있습니다.



```
val myVal1: Any = 10
val myVal2: Any? = myVal1

val myVal3: Any? = 10
val myVal4: Any = myVal3 //error
val myVal5: Any = myVal3 as Any
```

```
val myInt1: Int = 10
val myInt2: Int? = myInt1

val myInt3: Int? = 10
val myInt4: Int = myInt3 //error
val myInt5: Int = myInt3 as Int
```

## Null Safety – 일치 연산자

---

- ==을 structural equality, ===은 referential equality
- ==은 값에 대한 비교이고 ===은 객체에 대한 비교
- 일반 객체인지, 기초 데이터 타입의 객체인지에 따라 차이
- ? 에 의해 nullable로 선언되었는지에 따라 차이
- 일반 객체들은 ?에 의해 nullable로 선언

```
val user4=User()  
val user5: User?=user4  
println("user4==user5 is ${user4==user5}") //true  
println("user4===user5 is ${user4===user5}") //true
```

# Null Safety – 일치 연산자

---

- 기초 타입의 객체

```
val data3=1000
val data4=1000
val data5: Int?=1000
val data6: Int?=1000
println("data3==data4 is ${data3==data4}") //true
println("data3===data4 is ${data3===data4}") //true
println("data5==data6 is ${data5==data6}") //true
println("data5===data6 is ${data5===data6}") //false
```

- 기초 데이터 타입이 아닌 일반 클래스의 객체는 ==, ===의 차이가 없다. ?에 의해 boxing이 되든 안되든 동일 reference를 참조하면 true, 다른 객체이면 false
- 기초 데이터 타입의 변수 선언시 자바의 Wrapper 클래스(Integer 같은)를 직접 이용해 생성하면 객체가 생성되는 것이므로 ==은 값을, ===은 객체를 비교
- 기초 데이터 타입의 변수 선언시 Wrapper 클래스 이용없이 Int, Double 등으로 이용하면 ==, === 모두 값 비교
- ?에 의해 선언된 기초 데이터 타입의 변수는 내부적으로 boxing 되어 객체가 만들어 진다.

# Null Safety – 연산자

---

## Null 안전 관련 연산자

연산자	사용법	설명
?	val a: Int?	a 변수를 nullable로 선언
?:	A ?: B	A가 null이면 B 실행
?.	A?.length	A가 null이면 결과 값이 null, null이 아니면 length
!!	A !! B	A가 null 이 아닌경우만 B 실행. null이면 Exception 발생

# Null Safety – 연산자

---

## Null 확인 연산자 ?.

- 객체가 null 인지 체크한 후 null 이 아닌 경우에 한해서 멤버 접근

```
fun main(args: Array<String>) {  
    var data1: String? = "kkang"  
  
    val length1: Int? = if(data1 != null){  
        data1.length  
    } else {  
        null  
    }  
}
```

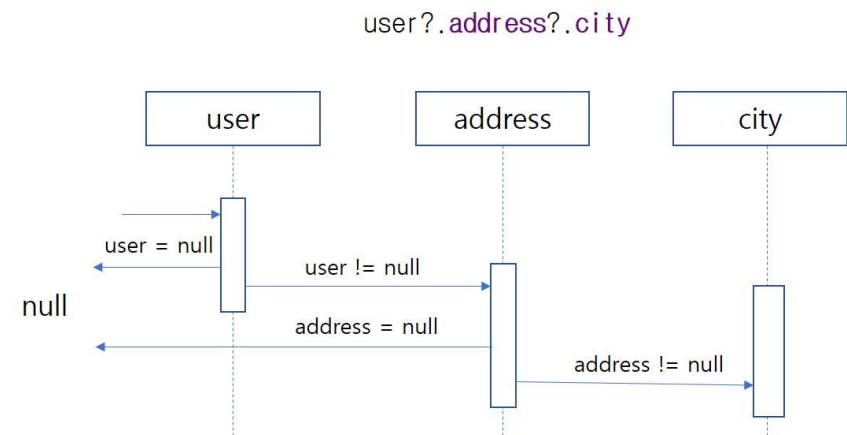
```
fun main(args: Array<String>) {  
    var data1: String? = "kkang"  
  
    var length2: Int? = data1?.length  
    println(length2)  
  
    data1=null  
    length2 = data1?.length  
    println(length2)  
}
```

# Null Safety – 연산자

## Null 확인 연산자 ?.

- Null 체크는 객체의 연결 구조에서도 사용이 가능

```
class Address {  
    val city: String?="seoul"  
}  
  
class User {  
    val address: Address? = Address()  
}  
  
fun main(args: Array<String>) {  
    val user: User? = User()  
  
    println(user?.address?.city)  
}
```



# Null Safety – 연산자

---

## Null 확인 연산자 ?.

- Null 이 아닌 경우 특정 구문이 수행되어야 하는 경우

```
fun main(args: Array<String>) {  
    val array= arrayOf("hello", null, "kkang")  
  
    array.forEach {  
        if(it != null){  
            println("$it .. ${it.length}")  
        }  
    }  
  
    array.forEach {  
        it?.let {  
            println("$it .. ${it.length}")  
        }  
    }  
}
```

# Null Safety – 연산자

---

## 엘비스 연산자 ?:

- Null 인 경우에 처리를 명시

```
fun main(args: Array<String>) {  
    var data: String? = "kkang"  
  
    var length: Int = if(data != null){  
        data.length  
    }else {  
        -1  
    }  
  
    data=null  
  
    length=data?.length ?: -1  
  
    println(length)  
  
    data ?: println("data is null")  
}
```

실행결과

-1

data is null

# Null Safety – 연산자

---

## 예외 발생 연산자 !!

- Null 인경우 Exception 을 발생시키기 위한 연산자

```
fun main(args: Array<String>) {  
    var data: String? = "kkang"  
  
    data!!.length  
  
    data=null  
  
    data!!.length  
}
```

### 실행결과

Exception in thread "main" kotlin.KotlinNullPointerException



# 감사합니다

단단히 마음먹고 떠난 사람은  
산꼭대기에 도착할 수 있다.  
산은 올라가는 사람에게만 정복된다.



윌리엄 셰익스피어  
William Shakespeare