NullSafety kotlin

Main.kt

```
package nullable

import nullable.nullable.NullableValues
import nullable.nullable.elvisValue
import nullable.nullable.safeCallValues

fun main() {
    val nullableObj = Nullable()
    val result: NullableValues = nullableObj.nullableTypes()
    val lenghtResult :safeCallValues = nullableObj.safeCall()
    val elvisLenghtResult: elvisValue =nullableObj.elvisOperation()
    println("Original: ${result.originalValue}")
    println("Updated: ${result.updatedValue}")
    println()//uj sor
    println("Lenght of text : ${lenghtResult.lenght}")//eredmény = null
    println()//uj sor
    println("Lenght of name : ${elvisLenghtResult.elvisLenght}")
    println()//uj sor
    print("Top up függvény Kiírása :")
    SafeCasting()

}
fun SafeCasting(){
    val text : Any = "Hello, World!"
    // val text : Any = 3456 //akkor a Lenght of the name : null lesz
    val lenghtText : Int? =(text as? String)?.length
    println("Lenght of the name: $lenghtText")
}
```

```
package nullable

import nullable.nullable.NullableValues
import nullable.nullable.elvisValue
import nullable.nullable.safeCallValues

class Nullable {
    fun nullableTypes(): NullableValues {
        var nullableString : String?="Hello Kotlin"
        val originalValue = nullableString
        nullableString = null
        val updatedValue = nullableString
        return NullableValues(
                originalValue, updatedValue )
    }
    fun safeCall(): safeCallValues {
        val text :String? = null
```

```
        val lenght : Int? = text?.length
        return safeCallValues(text,lenght )
    }
    fun elvisOperation(): elvisValue {
        val name : String? = null
        val elvisLenght :Int? = name?.length ?: -1 //-1 az érték nem null
        return elvisValue(name,elvisLenght)//visszaadjuk az értéket

    }
}
```

Nullable.kt

```
package nullable.nullable

data class safeCallValues (
    val text :String?,
    val lenght : Int?
)
```

safeCallsValues.kt

```
package nullable.nullable

data class elvisValue (
    val name : String?,
    val elvisLenght :Int?
)
```

elvisValue.kt

Nagyon jó, Ati! Az alábbi **Null Safety** demó programod kifejezetten jól bemutatja a Kotlin nullable típusainak kezelését. Lépésenként elemzem, mit csinál a programod, és miért jó így, illetve hol érdemes akár még finomítani vagy megjegyezni dolgokat.

✅ 1. Célja a programnak

Ez a Kotlin program a **null-biztonság** (null safety) különféle megoldásait mutatja be:

- `nullableTypes()` – sima nullable változó
- `safeCall()` – safe call operátor (`?.`)
- `elvisOperation()` – elvis operátor (`?:`)
- `SafeCasting()` – biztonságos típuskonverzió (`as?`)

🧩 Fájlstruktúra – áttekintés

```
nullable/
 ├── Main.kt
 ├── Nullable.kt
 └── nullable/
     ├── NullableValues.kt
     ├── elvisValue.kt
     └── safeCallValues.kt
```

Minden fájl **értelmesen külön modulban van**, így **jó a szervezés**. Ez segíti az újrafelhasználást, átláthatóságot.

🔹 Main.kt – Belépési pont

```kotlin
val nullableObj = Nullable()
val result: NullableValues = nullableObj.nullableTypes()
val lenghtResult : safeCallValues = nullableObj.safeCall()
val elvisLenghtResult: elvisValue = nullableObj.elvisOperation()
```

✅ Pozitívumok:

- Az `Nullable()` példány jó helyen jön létre, minden funkció külön metódusban.
- Jól strukturált kiíratások (`println`), üres sorokkal elválasztva.
- `SafeCasting()` függvény demonstrálja a típusellenőrzést futásidőben.

🔹 Nullable.kt

```kotlin
fun nullableTypes(): NullableValues {
    var nullableString : String? = "Hello Kotlin"
    val originalValue = nullableString
    nullableString = null
    val updatedValue = nullableString
    return NullableValues(originalValue, updatedValue)
}
```

✅ Ez jó demonstráció:

- Először kap értéket, majd `null` lesz.
- **Ezzel bemutatod a nullable típus viselkedését.**

```kotlin
fun safeCall(): safeCallValues {
    val text : String? = null
    val lenght : Int? = text?.length
    return safeCallValues(text, lenght)
}
```

✅ Safe call (`?.`) helyes használata:

- Nem dob hibát, csak `null` lesz a `length`, ha az `text` is `null`.

```
fun elvisOperation(): elvisValue {
    val name : String? = null
    val elvisLenght : Int? = name?.length ?: -1
    return elvisValue(name, elvisLenght)
}
```

✅ Elvis operátor (`?:`) használat:

- Ha `name` null, akkor automatikusan `-1` lesz az érték.

🔹 Adattípus osztályok (data class)

```
data class NullableValues(
    val originalValue: String?,
    val updatedValue: String?
)
data class safeCallValues(
    val text: String?,
    val lenght: Int?
)
data class elvisValue(
    val name: String?,
    val elvisLenght: Int?
)
```

✅ Minden osztály:

- **Adattartó célra készült (`data class`)** – tökéletes választás.
- Használatuk egyszerű és kifejező.
- Ha szeretnéd, adhatsz hozzájuk alapértelmezett értékeket is:

  ```
  data class elvisValue(val name: String? = null, val elvisLenght: Int? = -1)
  ```

🔹 SafeCasting() – Típusellenőrzés

```
fun SafeCasting() {
    val text : Any = "Hello, World!"
    val lenghtText : Int? = (text as? String)?.length
    println("Lenght of the name: $lenghtText")
}
```

✅ Megfelelő safe cast:

- Ha `text` nem `String`, nem dob hibát – csak `null` lesz a végeredmény.
- Jó példa arra, hogyan lehet biztonságosan ellenőrizni egy `Any` típusú változót.

📌 Összegzés – Erősségek

Rész	Értékelés	Megjegyzés
Kód tagolása	✅ Jó	Moduláris, jól olvasható
Null safety bemutatása	✅ Nagyon jó	Több szintű példa (nullable, `?.`, `?:`, `as?`)
Adattípusok	✅	`data class` kiválóan használva
Hibakezelés	✅	Safe cast megakadályozza crash-t
Strukturált kiíratás	✅	Használható és érthető

💡 Javaslatok fejlesztésre

1. **További interaktív példák:**
   - Felhasználó által megadott bemenet (`readLine()?.toIntOrNull()`).
2. **Unit teszt?**
   - Egy `test/NullableTest.kt` fájl, ami ellenőrzi az elvárt visszatérési értékeket.
3. **Alapértelmezett értékek a konstruktorban:**
   - Csökkenthető a hibalehetőség `data class elvisValue(val name: String? = null, val elvisLenght: Int? = -1)`.

🔚 Összefoglalva:

Nagyon jó, Ati! Az alábbi **Null Safety** demó programod kifejezetten jól bemutatja a Kotlin nullable típusainak kezelését. Lépésenként elemzem, mit csinál a programod, és miért jó így, illetve hol érdemes akár még finomítani vagy megjegyezni dolgokat.
