Kotlin – Résumé sur la nullabilité

Principe fondamental (Null-safety)

- En Kotlin, **les variables ne sont PAS nullables par défaut**.
- Objectif : éviter les `NullPointerException` **à la compilation**.

```
var nom: String = "Sak"   // non-null
// nom = null ❌
```

Le `?` — Autoriser explicitement `null`

- Le `?` fait **partie du type**.
- Indique que la variable **peut être null**.

```
var nom: String? = "Sak"
nom = null // ✅
```

Conséquence

- Accès direct interdit, car la valeur **pourrait être null**.

```
// nom.length ❌
```

Accéder à une variable nullable (bonnes pratiques)

1️⃣Safe call `?.`

```
nom?.length
```

- Si `nom != null` → valeur - Si `nom == null` → retourne `null`

2️⃣Elvis operator `?:`

```
val longueur = nom?.length ?: 0
```

- Fournit une valeur par défaut si `null`.

3️⃣Test explicite (smart cast)

```
if (nom != null) {
    nom.length // OK
}
```

Le `!!` — Forcer le non-null (dangereux)

- Appelé **Non-null assertion operator**.
- Dit au compilateur : *« fais-moi confiance »*.

```
nom!!.length
```

⚠️Si `nom == null` → **NullPointerException**.

Bonnes et mauvaises pratiques

❌À éviter

```
findViewById<TextView>(R.id.title)!!
```

- Supprime la sécurité de Kotlin.

✅Alternatives

- `?.`
- `?:`
- ViewBinding
- `lateinit` (si approprié)

Tableau récapitulatif

Élément	Signification	Risque
`String`	Non-nullable	Aucun
`String?`	Nullable	Doit être géré

Élément	Signification	Risque
`?.`	Accès sécurisé	Aucun
`?:`	Valeur par défaut	Aucun
`!!`	Forcer non-null	NPE

Idée clé à retenir

> `?` **= je reconnais l'incertitude**
> `!!` **= je prends la responsabilité**

En Android, privilégier `?.` **et** `?:`, éviter `!!`.