Fogalom	Mi ez?	Mire való?
`interface`	Egy **szerződés**, amit az osztályok betartanak	Csak **metódusokat (függvényeket)** tartalmaz, nincs adattag
`constructor`	Egy **különleges függvény**, ami egy osztály **példányosításakor** fut le	Általában **adattagokat** inicializál (pl. `val name: String`)

Main.kt

```
package interface2

fun main() {

    val circle = Circle()
    val  coloredCircle =ColoredCircle()
    val isFilled=coloredCircle.isFilled()


    circle.draw()
    println()
    coloredCircle.draw()
    println()
    coloredCircle.fill()
    println()
    println("Is filled ? : $isFilled")
}
```

interface

```
package interface2

interface Fillable {
    fun fill()
    fun isFilled(): Boolean{
        return true
    }
}
```

ColoredCircle osztály

```
package interface2
// öröklödés vagy kötelezövé teszük az interface tulajdonságait
//osztély : Shape,Fillable
class ColoredCircle : Shape,Fillable {
    override fun fill() {
        println("Filling the circle with color ")
    }
    override fun draw() {
        println("Drawing a colored Circle")
    }
}
```

Shape interface

```
package interface2

interface Shape {
    fun draw()
}
```

Circle osztály

```
package interface2
// öröklödés vagy kötelezövé teszük az interface tulajdonságait
//osztály azaz class Circle : Shape
class Circle : Shape { //hogy felajalja az implements membert az interfacenek
    //nem szabad absztraknak lennie tehát
    //fun draw() nem kell legyen {} -ez a kapcsos zárojel
    override fun draw() {
        println("Drawing a circle")
    }
}
```

Az osztályok öröklik az interfacek tulajdonságát

🧩 Az osztályok öröklik az interface-ek tulajdonságait...

...**az `override` kulcsszóval implementálva a metódusaikat.**
➡️ Vagyis: nem "öröklik" szó szerint, hanem **kötelező megvalósítaniuk (implementálniuk)**, amit az interface előír.

✅ Mikor mit használunk?

Használat	Mit használunk?	Mikor használjuk?
Közös viselkedés megosztására több osztály között	`interface`	Ha **különböző osztályoknak** kell ugyanazt a **függvényt (pl. draw)** megvalósítaniuk
Példányosításkor adatok beállítására	`constructor`	Ha egy osztály példányát úgy akarjuk létrehozni, hogy **adunk neki adatokat** (pl. `val name: String`)
Az objektum indulásakor végrehajtandó kódhoz	`init` blokk	Ha valamit **azonnal futtatni akarunk a példány létrejöttekor** (pl. naplózás, validálás stb.)

🔍 Példák

1. Interface használata

```
interface Drawable {
    fun draw()
}
class Circle : Drawable {
    override fun draw() {
        println("Drawing a circle")
    }
}
```

📌 **Mikor hasznos?**

- Ha azt akarod, hogy **különböző típusú formák** (pl. `Circle`, `Square`, `Triangle`) mind tudják a `draw()` műveletet – de **nem egy osztályból származnak**.

---

2. Constructor használata

```
class Dog(val name: String) {
    fun bark() {
        println("$name is barking!")
    }
}
val dog = Dog("Buddy") // <== konstruktor használat
```

📌 **Mikor hasznos?**

- Ha például egy kutyának **nevet kell adnunk**, és azt akarjuk, hogy minden kutya rendelkezzen vele.

---

3. Constructor + init blokk

```
class User(val name: String) {
    init {
        println("User $name created")
    }
}
val user = User("Ati")
```

📌 **Mikor hasznos?**

- Ha valamit **el akarunk végezni azonnal**, amikor az objektum létrejön (pl. logolás, ellenőrzés, üdvözlő üzenet).

🧠 Összefoglalva:

Használat célja	Mit használj?
Több osztályra közös művelet	`interface`
Objektum létrehozás paraméterekkel	`constructor`
Objektum indulásakor kód futtatása	`init` blokk
Viselkedések kombinálása	`interface + constructor`

Az **absztrakt** (angolul: *abstract*) egy **terv**, egy **vázlat**, ami **nincs teljesen kidolgozva**.
Az absztrakt dolgokat **nem lehet közvetlenül példányosítani**, mert **nincs bennük minden véglegesítve**.

Kotlinban: `abstract` kulcsszó

- Egy **absztrakt osztály**: olyan osztály, amelyet **nem lehet példányosítani közvetlenül**.
- Egy **absztrakt függvény**: olyan függvény, aminek **nincs törzse** (`{}`), tehát a viselkedését **al-osztályban kell megírni**.

```
abstract class Animal {
    abstract fun makeSound() // nincs törzse => kötelező felülírni
}

class Dog : Animal() {
    override fun makeSound() {
        println("Vau!")
    }
}

fun main() {
    val dog = Dog()
    dog.makeSound()  // "Vau!"
    // val a = Animal() // ❌ Hibás! Nem lehet példányosítani absztrakt osztályt
}
```

példa2

```
abstract class Vehicle {
    abstract fun startEngine()
}

class Car : Vehicle() {
    override fun startEngine() {
        println("Autó motorja elindult")
    }
}
```

Zárójel típusa	Név	Mire használjuk?	Példa
`()`	**Kerek zárójel**	**Függvényhívás**, **konstruktor**, paraméterlista	`print("Hello")`, `Person("Ati")`
`{}`	**Kapcsos zárójel**	**Kódtömb**, **függvénytörzs**, **osztálytest**, **lambda**	`{ println("Hello") }`
`[]`	**Szögletes zárójel**	**Tömb vagy lista elérés**, **indexelés**	`myList[0]`, `array[1]`