```
package oops
import inheritance.Budgie
import inheritance.Dog
fun main() {
    //Példányosítás
    val myDog = Dog("Buddy")
    myDog.speak()
    myDog.wagTail()
    println()
    val myBudgies = Budgie("Prüntyi")
    val myBudgies2 = Budgie("Pakito")
    myBudgies.speak()
    myBudgies2.speak()
    myBudgies.budgietail()
    myBudgies2.budgietail()
}
```

Main.kt

```
package inheritance

open class  Animal(val name :String) {

    open fun speak(){
        println("$name makes a sound")
    }
}
```

Animal.kt

```
package inheritance

class Dog (name :String) :Animal(name){

    fun wagTail(){
        println("$name wags tail")
    }
    override fun speak(){ //Felülírtuk a kutyánk nevét
        println("$name bark")
    }
}
```

Dog.kt

```
package inheritance

class Budgie(name: String) : Animal (name){
    fun budgietail(){
        println("$name csipcsip Tail")
    }
}
```

Budgie.kt

Nagyon jó úton haladsz, Ati! 🎉 Amit írtál, az egy **szép és jól strukturált példája a Kotlin öröklődésnek (inheritance)**, `open` osztállyal, felülírt metódussal (`override`), és saját egyedi metódusokkal. Most írok róla egy rövid, **érthető elemzést**, hogy jobban rögzüljön, mit csináltál:

🔁 Kotlin Öröklődés – Magyarázat a kódodra

📦 Alap `Animal` osztály

```
open class Animal(val name: String) {
    open fun speak() {
        println("$name makes a sound")
    }
}
```

- `open class`: csak így lehet örökölni belőle.
- `open fun speak()`: ez teszi lehetővé, hogy a `speak()` metódust a leszármazott osztályok `override`-olják.

🐶 `Dog` osztály öröklés

```
class Dog(name: String) : Animal(name) {
    fun wagTail() {
        println("$name wags tail")
    }

    override fun speak() {
        println("$name bark")
    }
}
```

- A `Dog` az `Animal`-ből örököl.
- Felülírja a `speak()` metódust, így mást mond, mint az alap `Animal`.
- Van egy saját metódusa: `wagTail()`.

🐦 `Budgie` osztály öröklés

```
class Budgie(name: String) : Animal(name) {
    fun budgietail() {
        println("$name csipcsip Tail")
    }
}
```

- Szintén az `Animal`-ből örököl.
- Nem írja felül a `speak()`-et, tehát ugyanazt használja, amit az `Animal` ad.
- Van egy saját metódusa: `budgietail()`.

🧪 `main()` – példányosítás, metódushívások

```
val myDog = Dog("Buddy")
myDog.speak()     // --> Buddy bark
myDog.wagTail()   // --> Buddy wags tail

val myBudgies = Budgie("Prüntyi")
val myBudgies2 = Budgie("Pakito")
myBudgies.speak()     // --> Prüntyi makes a sound
myBudgies2.speak()    // --> Pakito makes a sound
myBudgies.budgietail() // --> Prüntyi csipcsip Tail
```

✅ Minden példány jól működik, és külön metódusokat hívnak attól függően, hogy `Dog` vagy `Budgie`.

🧠 Összefoglalás – Amit megtanultál:

Fogalom	Jelentés
`open class`	Örökölhető osztály.
`open fun`	Felülírható metódus.
`override fun`	Egy metódus felülírása.
Konstruktorhívás	`: Animal(name)` szintaxissal történik.
Egyedi metódus	`budgietail()` és `wagTail()` nem léteznek az `Animal`-ban, csak az adott osztályban.

Másik példa:

```
package Engine

fun main(){

    val myCar= Car("Toyota","Canry")
    myCar.start()
}
```

```
package Engine

open class Vehicle(val brand:String) {
    open fun start(){
        println("Starting the $brand vehicle ")
    }
}
```

```
package Engine

class Car (brand: String, val model:String): Vehicle(brand){
    override fun start() {
        super.start()
        println("Car is ready to go")
    }
}
```

🧱 Részletes magyarázat

📦 `Vehicle` (ősosztály)

```
open class Vehicle(val brand: String) {
    open fun start() {
        println("Starting the $brand vehicle")
    }
}
```

- `open class`: csak így lehet örökölni.
- `open fun start()`: ez teszi lehetővé, hogy a gyerekosztály (`Car`) felülírja.
- `brand`: konstruktorparaméter, amit a `Car` is örököl.

🚗 `Car` (leszármazott osztály)

```
class Car(brand: String, val model: String) : Vehicle(brand) {
    override fun start() {
        super.start() // meghívja az ősosztály start() függvényét
        println("Car is ready to go")
    }
}
```

- Örökli a `Vehicle` osztályt.

- Átadja a `brand`-et a `Vehicle` konstruktorának (`: Vehicle(brand)`).
- Hozzáadja a saját mezőjét: `model`.
- A `start()` metódust felülírja, de **megtartja az ősosztály működését is**, mert hívja a `super.start()`-ot.

▶️ `main()` függvény

```
fun main() {
    val myCar = Car("Toyota", "Camry")
    myCar.start()
}
```

Kimenet:

```
Starting the Toyota vehicle
Car is ready to go
```

✅ Először meghívja az `Vehicle.start()`-ot (`super.start()`), majd folytatja a `Car` saját üzenetével.

🧠 Amit ez a példa tanít

Fogalom	Jelentés
`super.start()`	Meghívja az ősosztály azonos nevű metódusát.
Konstruktor öröklés	A `Car` átadja a `brand` értéket az `Vehicle`-nak.
Osztály + metódus öröklés	A `Car` örökli a `start()` metódust, amit testreszab.

💡 Tipp

Ha szeretnél például a `model` értékre is utalni, bővítheted a `start()`-ot:

```
override fun start() {
    super.start()
    println("$brand $model is ready to go")
}
```
