Main.kt

```
package functionalOperations

fun main(){

    val myMap = FunctionalOperations()
    val functionalOperations :ResultData = myMap.mapOperations()
    val filteredNumbers : Filterresult = myMap.filterOperations()


    println("Original Numbers : ${functionalOperations.original}")
    println("Squared Numbers : ${functionalOperations.squared}")
    println()
    println("All numbers : ${functionalOperations.original.joinToString()}" +
            ", ${functionalOperations.squared.joinToString()}")
    println()
    println("Filtered Numbers : ${filteredNumbers.filtered} ")
    println()
    reduceOperations() //kiiratjuk a top-up függvényt
}
//Top-up függvény
//Ha nem egy osztályban van hanem egy kódban mondjuk a fun main alatt felett
//akkor csak abból a fájlból / osztályból hívható meg a class jobb módszer
//de ha gyorsan kell jó a top-up függvény is.
fun reduceOperations() {
    val numbers:List<Int> = listOf(1,2,3,4,5)
    val sum = numbers.reduce{acc:Int, number:Int -> acc + number}
    //reduce végig megy a listán balról jobbra
    //és összeadjuk vele a számokat
    println("Original numbers : $numbers")
    println("Sum : $sum")
}
```

ResultData.kt

```
package functionalOperations

data class ResultData (
    val original: List<Int>,
    val squared: List<Int>
)
data class Filterresult (
    val original: List<Int>,
    val filtered: List<Int>
)
```

```
package functionalOperations
class FunctionalOperations {
    fun mapOperations(): ResultData { //ResulData osztály elvégzi az értékek
visszaadását
        val numbers: List<Int> = listOf(1, 2, 3, 4, 5)
```

```
        val squaredNumbers: List<Int> = numbers.map { it * it }//it*it négyzetre
emel
        //visszaadjuk a ResulDatának
        //így 2 osztály lesz elérhető a Main-ben is.
        return ResultData (
            original = numbers,
            squared = squaredNumbers
        )
    }
    fun filterOperations() :Filterresult {
        val numbers: List<Int> = listOf(1,2,3,4,5)
        val eventNumbers :List<Int> = numbers.filter { it % 2 == 0 }

        return Filterresult(
            original = numbers,
            filtered = eventNumbers
        )
    }
}
```

elemzés:

Elemzés a programról

A program célja, hogy bemutassa a Kotlin nyelv **funkcionális műveleteinek** (map, filter, reduce) használatát egy egyszerű példán keresztül. A program több kisebb részből áll, amelyek különböző műveleteket végeznek el egy egész számokat tartalmazó listán.

---

1. `FunctionalOperations` osztály

Ez az osztály két fő függvényt tartalmaz:

- `mapOperations()`
  - Bemenet: nincs paramétere, egy fix lista (`numbers`) értékeit használja.
  - Művelet: a `numbers` lista elemeit négyzetre emeli a `map` függvénnyel (`it * it`).
  - Kimenet: egy `ResultData` típusú objektumot ad vissza, amely tartalmazza az eredeti és a négyzetre emelt számokat is.
- `filterOperations()`
  - Bemenet: szintén a fix `numbers` lista.
  - Művelet: kiszűri a páros számokat a `filter` függvény segítségével (`it % 2 == 0`).
  - Kimenet: egy `Filterresult` típusú objektumot ad vissza, amely tartalmazza az eredeti és a szűrt (páros) számokat.

---

2. Adatmodellek: `ResultData` és `Filterresult`

- `ResultData`
  Egy adatosztály (`data class`), amely két listát tárol: az eredeti számokat és azok négyzetét.
- `Filterresult`
  Egy adatosztály, amely az eredeti számok listáját és a szűrt (páros) számok listáját tárolja.

Ezek az adatmodellek megkönnyítik az eredmények átlátható és strukturált visszaadását.

---

3. `main()` függvény

- Létrehoz egy `FunctionalOperations` példányt.
- Meghívja a `mapOperations()` és `filterOperations()` függvényeket, és eltárolja az eredményeket.
- Kiírja az eredeti számokat, a négyzetre emelt számokat, illetve a szűrt számokat.
- Meghívja a `reduceOperations()` függvényt, amely egy további műveletet mutat be.

---

4. `reduceOperations()` top-level függvény

- Egy listán végighalad a `reduce` funkcióval, amely összeadja a lista elemeit.
- Kiírja az eredeti számokat és az összegüket.
- A `reduce` függvény működése: balról jobbra haladva halmozza az értékeket az `acc` változóban.

Program általános jellemzői és erősségei

- **Funkcionális programozási stílus**: `map`, `filter`, `reduce` használata nagyon tiszta, Kotlin-hoz illő, idiomatikus megoldás.
- **Átlátható kód**: az adatmodellek és az osztályok jól elkülönítik az egyes részeket.
- **Könnyű bővíthetőség**: további hasonló műveletek könnyen beilleszthetők.
- **Használható oktatási példa**: egyszerű és jól demonstrálja a funkcionális műveletek lényegét.

Lehetséges fejlesztések

- A `numbers` lista ne legyen minden metóduson belül duplikálva, hanem legyen osztályszintű vagy paraméterként átadott.
- `Filterresult` osztályban az elnevezés `eventNumbers` helyett helyesen `evenNumbers` (páros számok) lenne.
- Több komment, főleg angol-magyar egységesítése.
- Hibakezelés, ha nagyobb, komplexebb műveletek lennének.

Összességében ez a program nagyon jól szemlélteti a Kotlin funkcionális műveleteinek használatát egyszerű listákon, és egy jól szervezett struktúrát biztosít a különböző adattranszformációk visszaadásához.