```
package chainig


//top up függvények
fun chainingMap(): List<String> {
    val persons= listOf(

        Person("Alice",25),
        Person("Bob", 30),
        Person("Charlie", 22),
        Person("David", 35),
        Person("Eva", 28)
    )
    return persons.filter { it.age>25 }.map { it.name }
}

fun main(){
val book = Book()
val result :bookResult = book.chainingFlatMap()
    val namesOlderThan25 :List<String> = chainingMap()

    val product: Double = Product().allProduct().totalPriceAfterDiscount
    val allProduct = Product().allProduct().products.joinToString("\n")
{"${it.name} - ${it.Price} $"}
    println("This is all product : \n$allProduct")
    println("The Total is : $product $ dollar.")
    println()
    println(result.books.joinToString("\n"))
    println()
    println(result.distinctAuthors)
    println()
    println("Names of person older than 25: $namesOlderThan25")

}
```

main.kt

Book.kt

```
package chainig

class Book {
    fun chainingFlatMap(): bookResult {
        val books= listOf(
            dataBook("Kotlin in action", listOf("Dimitry Jemrov","Svetlana
Isakova")),
            dataBook("Effective Java", listOf("Joshua Bloch")),
            dataBook("Clean Code", listOf( "Dimitry Jemerow","Svetlana
Isakova"))
        )
        val distinctAuthors=books.flatMap { it.authors }.distinct()
```

```
        return bookResult(
            distinctAuthors = distinctAuthors,
            books = books
        )
    }
}
```

---

bookResult.kt

```
package chainig

class bookResult (
    val distinctAuthors : List<String>,
    val books : List<dataBook>
)
```

---

dataBook.kt

```
package chainig

data class dataBook (val title :String, val authors :List <String> )
```

---

dataproduct.kt

```
package chainig

data class dataProduct(val name: String, val Price: Double)
```

---

dataProductResult.kt

```
package chainig

class dataProductResult (
    val totalPriceAfterDiscount : Double =0.0,
    val products: List<dataProduct>
)
```

Person.kt

```
package chainig

data class Person (val name :String,val age :Int)
```

Product.kt

```
package chainig

class Product {
    fun allProduct(): dataProductResult { //
        var products :List<dataProduct> = listOf(
            dataProduct("Laptop", 1200.0),
            dataProduct("Mouse", 30.0),
            dataProduct("Keyboard", 80.0),
            dataProduct("Laptop Bag", 50.0)
        )
        var totalPriceAfterDiscount :Double =products.filter { it.Price>50 }.map
{ it.Price*0.9 }
            .sum()
        return dataProductResult (
                products = products,
                totalPriceAfterDiscount = totalPriceAfterDiscount
        )
    }
}
```

Nézzük részletesen:

✅ 1. Projekt struktúra

Package: `chainig`
Fájlok:

- `main.kt`
- `Book.kt`, `bookResult.kt`, `dataBook.kt`
- `Product.kt`, `dataProductResult.kt`, `dataProduct.kt`
- `Person.kt`

✅ 2. `main.kt` logikája

```
val book = Book()
val result: bookResult = book.chainingFlatMap()
val namesOlderThan25: List<String> = chainingMap()
```

Itt:

- `Book().chainingFlatMap()` visszaad egy `bookResult` objektumot.
- `chainingMap()` kiszűri az idősebb embereket és visszaadja nevüket (ha `age > 25`).
- A `Product()` példány létrehozza a terméklistát, kiszámítja a kedvezményes összárat.
- `joinToString("\n") { ... }` szépen formázza a terméklistát megjelenítéshez.

✅ 3. `Book.kt` és `bookResult.kt`

```
fun chainingFlatMap(): bookResult
```

- `flatMap`: kilapítja a szerzőlistát → `List<List<String>> → List<String>`
- `distinct()`: csak az egyedi neveket tartja meg.
- Ezt visszarakja egy `bookResult` nevű osztályba.

📌 Jó gyakorlat: A `bookResult` külön osztályban van → ez **tiszta adatmodell** és jól szétválasztott.

✅ 4. `Product.kt` és kapcsolódó osztályai

```
fun allProduct(): dataProductResult
```

- A `Product` osztály létrehoz egy `List<dataProduct>`-et.
- Kedvezmény: csak azok a termékek kapnak 10% kedvezményt, ahol `Price > 50`.
- Ezeket összegzi `sum()`-mal.
- Eredmény: `dataProductResult`, amely egy `totalPriceAfterDiscount` + `products`.

📌 Jól strukturált, könnyen bővíthető.

✅ 5. `dataBook`, `dataProduct`, `Person` osztályok

Mindhárom `data class` formában van deklarálva:

```
data class dataBook(val title: String, val authors: List<String>)
data class dataProduct(val name: String, val Price: Double)
data class Person(val name: String, val age: Int)
```

✅ Ez nagyon jó, mert automatikusan kapsz:

- `toString()`
- `equals()`, `hashCode()`
- `copy()`

Ezért működik a `joinToString` anélkül, hogy külön `toString()` metódust kéne írnod.

✅ 6. A `main()` output formázása

```
val allProduct = Product().allProduct().products
    .joinToString("\n") { "${it.name} - ${it.Price} $" }
```

Ez egy szép, olvasható listaformázás. Pl.:

```
Laptop - 1200.0 $
Mouse - 30.0 $
...
```

✅ Összefoglalás / Értékelés

Szempont	Értékelés	Megjegyzés
Struktúra	⭐⭐⭐⭐⭐	Kiválóan szétválasztott modulok
Kód tisztaság	⭐⭐⭐⭐☆	Jól olvasható, érthető
OOP + FP keverése	⭐⭐⭐⭐⭐	`data class`, `map`, `filter`, `flatMap`
Fejleszthetőség	⭐⭐⭐⭐☆	Könnyen bővíthető, de lehetne DI
Névhasználat	⭐⭐⭐⭐☆	`dataBook`, `dataProduct` → lehetne PascalCase

🔧 Javaslatok a továbbfejlesztésre:

1. **Kis refaktorálás**
   - `dataBook` → `DataBook`
   - `dataProduct` → `DataProduct`
   - `bookResult` → `BookResult`
   - Ez a Kotlin stílushoz jobban illik (PascalCase).
2. **Interface bevezetés (ha bővítenéd):**
   - Pl. `Productable`, `BookProvider` stb.
3. **Teszt hozzáadása**
   - Írj `test` mappába egy `BookTest.kt`-t pl. JUnit-tel.
