Main.kt

```
import dataClasses.Book
import dataClasses.Person

fun main() {
    //példányosítás
    val person = Person("Alice", 25)

    println("Name : ${person.name}")
    println("Age : ${person.age}")

    println("person details : $person")
    //tooStringel
    println("person details : ${person.toString()}")
    println()//uj sor
    println("Book_Info:")
    //pédányosítás
    val book = Book("Kotlin in action","Dimitry Jemerov", 2017)
    val updateBook = book.copy(year = 2022)// copy függvény
    val (title , author, year) = updateBook

    println("Original book : $book")
    println("Updated book : $updateBook")

    println("Desctructing - Title : $title, Author : $author, Year : $year")
}
```

Book.kt

```
package dataClasses

data class Book(val title :String, val author :String,val year :Int )
```

Person.kt

```
package dataClasses

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

✅ Mit csinál ez a kód?

1. **Példányosítasz egy `Person` és egy `Book` adatobjektumot**
2. Kiírod az értékeiket
3. Használod a `copy()` függvényt
4. Használod a **destrukturálást** (pl. `val (title, author, year) = book`)

5. A `toString()` automatikusan működik `data class`-eknél

💡 Mi az a `data class`?

Fogalom	Jelentés / Előny
`data class`	Olyan osztály, amit **főleg adatok tárolására** használunk
Automatikusan létrejönnek benne:	
`toString()`	Szép kiírás emberi olvasásra (`println(book)`)
`copy()`	Másolat készítése egy-két mező megváltoztatásával
`equals()` / `hashCode()`	Két példány összehasonlítása érték szerint
Destrukturálás	Könnyen "szétszedhető" az osztály (`val (x,y,z)=obj`)

🧠 Példa `copy()` használatra

```
val book = Book("Kotlin", "Dimitry", 2017)
val updatedBook = book.copy(year = 2022)
```

Ez olyan, mintha ezt írnád:

```
val updatedBook = Book("Kotlin", "Dimitry", 2022)
```

🔧 Mi az a destrukturálás?

```
val (title, author, year) = book
```

Ez **kibontja** a `book` adatait külön változókba:

- `title = book.title`
- `author = book.author`
- `year = book.year`

🧪 Kipróbált funkciók a kódodban:

Funkció	Kód
`data class`	`data class Book(...)`
`copy()`	`val updatedBook = book.copy(year = 2022)`
`toString()`	`println(book)` vagy `println(book.toString())`
Destructuring	`val (title, author, year) = book`

Nagyon jó kérdés, Ati! Nézzük a **különbséget az** `open class` **és a** `data class` **között** Kotlin nyelvben, röviden és érthetően:

🔷 `data class` – adatmodellezésre

Jellemző	Magyarázat
Csak **adatok tárolására** való	Olyan osztály, amiben **nincs vagy kevés logika**, csak adatok
Automatikus metódusok	`toString()`, `equals()`, `hashCode()`, `copy()`, `componentN()`
Használat célja	Adatok reprezentálása, pl. `User`, `Book`, `Product`, `Person`, stb.
Nem lehet `abstract`, `open`, `sealed`	Ezek **nem kombinálhatók** `data`-val
Öröklés: nem lehet belőle örökölni	De **implementálhat interfészt**

Példa:

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

🔷 `open class` – bővíthető osztály

Jellemző	Magyarázat
Lehetővé teszi az öröklést	Alapértelmezetten a Kotlin osztályok **final** (nem örökölhetők)
Nincs automatikus `toString` vagy `copy()`	Ezeket neked kell megírni, ha kell
Használat célja	Olyan osztály, aminek lesz **alaposztálya vagy leszármazottja**
Tartalmazhat logikát	Tartalmazhat viselkedést, például metódusokat, `init` blokkokat stb.

Példa:

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

🔑 Fő különbségek összefoglalva:

	`data class`	`open class`
Adatok reprezentálása	✅ Igen	❌ Nem kifejezetten
Örökölhető	❌ Nem	✅ Igen
Automatikus metódusok	✅ Igen (`copy`, `equals`, stb.)	❌ Neked kell megírni
Konstrukció célja	Adattárolás	Logikával bíró bővíthető osztály
`abstract/open` lehet?	❌ Nem	✅ Igen

Szuper kérdés, Ati! Nézzük meg **mikor használjuk a `data class`-t**, és **mikor az `open class`-t**, érthető példákkal:

🟢 `data class` – amikor *adatokat* szeretnél tárolni, modellezni

🔧 Mire való?

- Olyan osztály, ami csak **adatokat tartalmaz**
- **Nincs vagy alig van benne logika**
- Gyakran használod **adatátadásra**, **visszatérési értékként**, **JSON modellezésre**, stb.

📦 Példák:

- `User(name, age)`
- `Product(id, price)`
- `Book(title, author, year)`

✅ Használjuk:

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

💡 Mikor válaszd?

- Ha **egyszerű adathordozót** szeretnél
- Ha kell a `copy()`, `equals()`, `toString()` automatikusan
- Ha az osztály nem fog öröklődni (nem kell `open`)

🔴 `open class` – amikor *örökölhető*, logikát tartalmazó osztályt szeretnél

🔧 Mire való?

- Alap (szülő) osztály készítésére, amit más osztályok örökölhetnek
- Ha az osztály **metódusokat, működést, logikát** tartalmaz
- Lehet `abstract`, lehet `interface`-ből is származó

🚗 Példák:

- `Vehicle`, amit `Car`, `Bike` örököl
- `Animal`, amit `Dog`, `Cat` örököl
- `Shape`, amit `Circle`, `Rectangle` bővít

✅ Használjuk:

```
open class Animal(val name: String) {
    open fun makeSound() {
        println("Animal sound")
    }
}
```

🔁 Összefoglalva:

Mire van szükséged?	Használj…
Csak **adatokat** tárolsz	`data class`
Kell a `copy()` / `toString()`	`data class`
Örököltetni akarod az osztályt	`open class`
Kell benne logika (függvényekkel)	`open class`
Alaposztályként fog szolgálni	`open class`
