```
data class Sender (val name :String)
data class Message(val contentMessage : String, val isRead: Boolean, val sender:
Sender)

fun main() {
    val messages :List<Message> = listOf(
        Message("Hy",false ,Sender("Julia")),
        Message("empty",true ,Sender("Anna")),
        Message("Please....",false ,Sender("Kitty")),
        Message(" ",false ,Sender("George")),
        Message("I love you",true ,Sender("Anna")),
        Message("Please Help me",true ,Sender("Katty")),
        Message("Here your Money Thank you!",true ,Sender("Jennin"))

    )
    /*  - distinct () kiszűri a duplikált neveket
        - isNotBlank() ellenőrzi hogy nem -e üres stringről van szó
        - && mind a 2 feltételnek igaznak kell lennie
        - it.isRead olvasott Boolean ,!it.isRead nem olvasott Boolean (logikai
negalás) !
          az ellentéte a rendesnek
        - it. hivatkozunk a típusokra Kotlinban
        - sortedBy(Sender::name) Név szerint rakja a Senderel hivatkozunk a
típusra a data classban
          a ::name meg a névre Abc sorrrend szerint
        - map uj listát hozz létre az eredeti listából
    *  */

    val uniqueUnreadSenders :List<Sender> = messages
        .filter { it.contentMessage.isNotBlank() && !it.isRead }
        .map { it.sender }
        .distinct()
        .sortedBy(Sender::name)
    val uniqueReadSenders :List<Sender> = messages
        .filter { it.contentMessage.isNotBlank() && it.isRead}
        .map { it.sender }
        .distinct()
        .sortedBy(Sender::name)
    /*
    * uniqueUnreadSenders nem olvasott üzenetek nevek szerint
    * uniqueReadSenders olvasott üzenetek nevek szerint
    * */
    println("Nem olvasott üzenetek nevek szerint:")
          /*
           //for ciklus sender hivatkozik a Sender listára (in)
uniqueUnreadSenders - ben
           //majd printLine-val kiiratjuk az uj létrehozott sendert
           for (sender :Sender in uniqueUnreadSenders){
               println("${sender.name},")
           */
            /*  -.withIndex() egy listából indexelt sorozatot csinál
hogy,hozzáférj a for ciklusban
```

```
                az elemhez és az indexéhez.
               - Ha az `index` kisebb, mint `uniqueReadSenders.lastIndex`, akkor
kiírunk egy vesszőt utána.
               - ha nem else és nem írunk a.listIndex a lista utolsó eleme
            */

    for ((index, sender) in uniqueUnreadSenders.withIndex()){
            if (index < uniqueUnreadSenders.lastIndex)
                print("${sender.name}, ")
            else
                println(sender.name)//utolsó elemnél nincs vessző
        }
    println("Olvasott üzenetek: \n${uniqueReadSenders.joinToString (",\n") {
it.name } } ")
    //.joinToString {it.name} csak a neveket írja ki zárojel nélkül
    //(",\n") \n uj sor előtte , add hozzá
}
```

elemzés:

Adatmodellek

```
data class Sender (val name :String)
data class Message(val contentMessage : String, val isRead: Boolean, val sender:
Sender)
```

- **Sender**: csak egy mezője van: `name`, ami a feladó neve.
- **Message**: egy üzenetet ír le, amely három mezőt tartalmaz:
  - `contentMessage`: az üzenet szövege
  - `isRead`: boolean típus, ami azt mutatja, hogy az üzenet olvasott-e
  - `sender`: a feladó, azaz `Sender` típusú objektum

Üzenetlista

```
val messages :List<Message> = listOf(
    Message("Hy",false ,Sender("Julia")),
    Message("empty",true ,Sender("Anna")),
    Message("Please....",false ,Sender("Kitty")),
    Message(" ",false ,Sender("George")),
    Message("I love you",true ,Sender("Anna")),
    Message("Please Help me",true ,Sender("Katty")),
    Message("Here your Money Thank you!",true ,Sender("Jennin"))
)
```

Ez egy statikus lista, különféle üzenetekkel és különböző állapotokkal:

- Vannak **olvasott** és **nem olvasott** üzenetek.

- Vannak **üres** vagy **üres karaktereket** tartalmazó üzenetek (`" "` vagy `"empty"`), ezek kiszűrésre kerülnek.

Nem olvasott üzenetek szűrése

```
val uniqueUnreadSenders :List<Sender> = messages
    .filter { it.contentMessage.isNotBlank() && !it.isRead }
    .map { it.sender }
    .distinct()
    .sortedBy(Sender::name)
```

Mit csinál?

1. `filter { it.contentMessage.isNotBlank() && !it.isRead }`
    - Kiszűri azokat az üzeneteket, amik **nem üresek** és **nincsenek olvasva**.
2. `map { it.sender }`
    - A szűrt üzenetekből csak a feladókat (`Sender`) tartja meg.
3. `distinct()`
    - Megszünteti a duplikált `Sender` objektumokat (ugyanaz a név → ugyanaz az objektum).
4. `sortedBy(Sender::name)`
    - ABC sorrendbe rakja a feladókat név szerint.

Olvasott üzenetek szűrése

Ugyanez, csak az `isRead` érték **true**:

```
val uniqueReadSenders :List<Sender> = messages
    .filter { it.contentMessage.isNotBlank() && it.isRead}
    .map { it.sender }
    .distinct()
    .sortedBy(Sender::name)
```

Kiíratás – nem olvasottak nevei

```
println("Nem olvasott üzenetek nevek szerint:")
for ((index, sender) in uniqueUnreadSenders.withIndex()){
    if (index < uniqueUnreadSenders.lastIndex)
        print("${sender.name}, ")
    else
        println(sender.name) // utolsó elemnél nincs vessző
}
```

- `withIndex()` : lehetővé teszi az **index** és **elem** együttes elérését.
- **Feltétel**: ha **nem az utolsó elem**, akkor jön egy `,` a név után.
- Az **utolsó név** után nem tesz vesszőt, hanem csak simán új sort ír.

Kiíratás – olvasott üzenetek nevei

```
println("Olvasott üzenetek: \n${uniqueReadSenders.joinToString (",\n") { it.name } } ")
```

- `joinToString(",\n") { it.name }`: A `Sender` objektumokat név szerint kiírja, soronként elválasztva vesszővel:
  - Példa:

```
Anna,
Jennin,
Katty
```

Kimenet (elméletileg):

```
Nem olvasott üzenetek nevek szerint:
Julia, Kitty
Olvasott üzenetek:
Anna,
Jennin,
Katty
```

(George-t kiszűrtük, mert üres üzenetet küldött: `" "`)
(Anna kétszer is küldött üzenetet, de `distinct()` miatt csak egyszer jelenik meg.)

Összefoglalás – amit jól csináltál:

- 👍 `filter`, `map`, `distinct`, `sortedBy` helyes kombinálása
- 👍 `withIndex()` használat a pontos formázáshoz
- 👍 Szép kommentek, jól magyarázod a logikát
- 👍 A `data class`-okat jól alkalmazod