상태 저장하기 2

상태 저장하는 방법

- rememberSaveable는 UI 상태를 Bundle에 저장
  - 기본 데이터 타입은 자동으로 Bundle에 저장됨
  - 기본 타입이 아닌 경우 Custom Saver를 제공해야 함
- Custom Saver
  - Parcelize
  - listSaver
  - mapSaver
  - Custom Saver 클래스

```
@Composable
fun <T : Any> rememberSaveable(
    vararg inputs: Any?,
    saver: Saver<T, Any> = autoSaver(),
    key: String? = null,
    init: () -> T
): T
```

```
interface Saver<Original : Any?, Saveable : Any>
```

복원할 타입 저장할 타입

Parcelable & @Parcelize 이용

- @Parcelize를 사용하면 Parcelable 객체를 생성
  - App 수준의 build.gradle 파일에 플러그인 추가

```
plugins {
    id("kotlin-parcelize")
}
```

```
@Parcelize
data class City(val name: String, val country: String) : Parcelable

@Composable
fun CityScreen() {
    var selectedCity = rememberSaveable {
        mutableStateOf(City("Madrid", "Spain"))
    }
}
```

  - 주의 : 객체의 일부 상태만 변한다고 재구성되지 않음
    - **객체를 copy해서 객체 자체를 변경해야 재구성이 일어 남**

MapSaver

- mapSaver를 이용하여 값 집합으로 객체 변환

```
data class City(val name: String, val country: String)

val CitySaver = run {
    val nameKey = "Name"
    val countryKey = "Country"
    mapSaver(
        save = { mapOf(nameKey to it.name, countryKey to it.country) },
        restore = { City(it[nameKey] as String, it[countryKey] as String) }
    )
}

@Composable
fun CityScreen() {
    var selectedCity = rememberSaveable(stateSaver = CitySaver) {
        mutableStateOf(City("Madrid", "Spain"))
    }
}
```

***run 함수 : 블록 실행하고, 마지막 표현식 반환**

ListSaver

- listSaver를 이용하여 Index로 맵의 키를 대신해서 사용

```
data class City(val name: String, val country: String)

val CitySaver = listSaver<City, Any>(
    save = { listOf(it.name, it.country) },
    restore = { City(it[0] as String, it[1] as String) }
)

@Composable
fun CityScreen() {
    var selectedCity = rememberSaveable(stateSaver = CitySaver) {
        mutableStateOf(City("Madrid", "Spain"))
    }
}
```

Saver 함수

- 저장하는 방법 직접 지정하여 객체 저장

```
import androidx.compose.runtime.saveable.Saver

data class Holder(var value: Int)

// this Saver implementation converts Holder ob
// to Int which we can save
val HolderSaver = Saver<Holder, Int>(
    save = { it.value },
    restore = { Holder(it) }
)
```

Kotlin의 Scope functions

- Scope functions
  - 객체의 컨텍스트 내에서 코드 블록을 실행하는 것이 목적인 함수
    - 임시 스코프가 설정되고, 이 스코프 내에서 해당 객체의 이름 없이 멤버 접근이 가능
    - 코드를 간결하고 읽기 쉽게 만드는 역할 수행
  - 5개의 함수 : let, with, run, apply, also

```kotlin
data class Person(var name: String, var age: Int, var city: String)
    fun moveTo(newCity: String) { city = newCity }
    fun incrementAge() { age++ }
}

fun main() {
    val alice = Person("Alice", 20, "Amsterdam")
    println(alice)
    alice.moveTo("London")
    alice.incrementAge()
    println(alice)
}
```

```
Person(name=Alice, age=20, city=Amsterdam)
Person(name=Alice, age=21, city=London)
```

```kotlin
data class Person(var name: String, var age: Int, var city: String)
    fun moveTo(newCity: String) { city = newCity }
    fun incrementAge() { age++ }
}

fun main() {
    Person("Alice", 20, "Amsterdam").let {
        println(it)
        it.moveTo("London")
        it.incrementAge()
        println(it)
    }
}
```

```
Person(name=Alice, age=20, city=Amsterdam)
Person(name=Alice, age=21, city=London)
```

Kotlin의 Scope functions

- Function selection
  - 비슷한 기능을 수행하지만, 객체 참조 방식 및 리턴 값에 따라 선택

Function	Object reference	Return value	Is extension function
`let` ↗	`it`	Lambda result	Yes
`run` ↗	`this`	Lambda result	Yes
`run` ↗	-	Lambda result	No: called without the context object
`with` ↗	`this`	Lambda result	No: takes the context object as an argument.
`apply` ↗	`this`	Context object	Yes
`also` ↗	`it`	Context object	Yes

let

- Context object는 argument (it) 로서 이용가능, return value는 lambda result

```
inline fun <T, R> T.let(block: (T) -> R): R
```

  - 하나 이상의 함수를 호출 / null 체크 / local scope에서의 지역 변수 표현

```
val numbers = mutableListOf("one", "two", "three", "four", "five")
numbers.map { it.length }.filter { it > 3 }.let {
    println(it)
    // and more function calls if needed
}
```

```
[5, 4, 4]
```

```
val numbers = listOf("one", "two", "three", "four")
val modifiedFirstItem = numbers.first().let { firstItem ->
    println("The first item of the list is '$firstItem'")
    if (firstItem.length >= 5) firstItem else "!" + firstItem + "!"
}.uppercase()
println("First item after modifications: '$modifiedFirstItem'")
```

```
val str: String? = "Hello"
//processNonNullString(str)       // compilation error: str can be null
val length = str?.let {
    println("let() called on $it")
    processNonNullString(it)      // OK: 'it' is not null inside '?.let { }'
    it.length
}
```

```
let() called on Hello
```

with

- Context object는 receiver (this)로서 이용가능, return value는 lambda result

```
inline fun <T, R> with(receiver: T, block: T.() -> R): R
```

  - 반환된 결과가 필요 없는 함수 호출 / 객체의 속성 설정 및 속성 이나 함수가 값 계산에 사용되는 경우

```
val numbers = mutableListOf("one", "two", "three")
with(numbers) {
    println("'with' is called with argument $this")
    println("It contains $size elements")
}
```

```
'with' is called with argument [one, two, three]
It contains 3 elements
```

```
val numbers = mutableListOf("one", "two", "three")
val firstAndLast = with(numbers) {
    "The first element is ${first()}," +
    " the last element is ${last()}"
}
println(firstAndLast)
```

```
The first element is one, the last element is three
```

run

- Context object는 receiver (this)로서 이용가능, return value는 lambda result

```
inline fun <T, R> T.run(block: T.() -> R): R
inline fun <R> run(block: () -> R): R
```

  - 객체를 초기화 하고, 반환값 계산 / 식이 필요한 여러 문장의 블록을 실행

```
val service = MultiportService("https://example.kotlinlang.org", 80)

val result = service.run {
    port = 8080
    query(prepareRequest() + " to port $port")
}

// the same code written with let() function:
val letResult = service.let {
    it.port = 8080
    it.query(it.prepareRequest() + " to port ${it.port}")
}
```

```
val hexNumberRegex = run {
    val digits = "0-9"
    val hexDigits = "A-Fa-f"
    val sign = "+-"

    Regex("[$sign]?[$digits$hexDigits]+")
}

for (match in hexNumberRegex.findAll("+123 -FFFF !%*& 88 XYZ")) {
    println(match.value)
}
```

apply

- Context object는 receiver (this)로서 이용가능, return value는 return value는 object itself

```
inline fun <T> T.apply(block: T.() -> Unit): T
```

  - **객체의 초기화**

```
val adam = Person("Adam").apply {
    age = 32
    city = "London"
}
println(adam)
```

also

- Context object는 argument (it) 로서 이용가능, return value는 object itself

```
inline fun <T> T.also(block: (T) -> Unit): T
```

  - Context object를 인수로서 사용하는 동작
    - 객체의 속성을 변경하지 않고 사용하는 경우

```
val numbers = mutableListOf("one", "two", "three")
numbers
    .also { println("The list elements before adding new one: $it") }
    .add("four")
```

Log methods

- Log.v(“tag”,”message”)
  - Verbose : 개발중에만 사용하여 상세 정보 표시
- Log.i(“tag”,”message”)
  - Information : 일반 정보 표시
- Log.d(“tag”,”message”)
  - Debug : debug용 로그
- Log.w(“tag”,”message”)
  - Warning : 경고 표시
- Log.e(“tag”,”message”)
  - error : error용 로그

Resource ID 찾기

- 컴포저블 내에서 Resource ID 획득하는 방법

```
val context = LocalContext.current
```

- **Context**
  - **Application과 관련된 정보**
  - **시스템 함수**
- **LocalContext.current**
  - **현재 컴포넌트가 속한 Context (액티비티) 제공**

R.id.hat

```
val imgID = context.resources.getIdentifier(
                    “hat” ,
                    "id",
                    context.packageName)
```

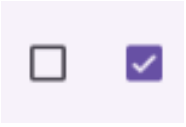
R.drawable.hat

```
val imgR = context.resources.getIdentifier(
                    “hat” ,
                    “drawable",
                   context.packageName)
```

CheckBox

- 사용자가 하나 이상의 항목을 선택할 때 사용

```
@Composable
fun Checkbox(
    checked: Boolean,
    onCheckedChange: ((Boolean) -> Unit)?,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    colors: CheckboxColors = CheckboxDefaults.colors(),
    interactionSource: MutableInteractionSource? = null
): Unit
```



```
val (checkedState, onStateChange) = remember { mutableStateOf(true) }
Row(
    Modifier
        .fillMaxWidth()
        .height(56.dp)
        .toggleable(
            value = checkedState,
            onValueChange = { onStateChange(!checkedState) },
            role = Role.Checkbox
        )
        .padding(horizontal = 16.dp),
    verticalAlignment = Alignment.CenterVertically
) {
    Checkbox(
        checked = checkedState,
        onCheckedChange = null // null recommended for accessibility w
    )
    Text(
        text = "Option selection",
        style = MaterialTheme.typography.bodyLarge,
        modifier = Modifier.padding(start = 16.dp)
    )
}
```

수고하셨습니다.