Лабораторийн ажил 2

Unit 2: Building App UI

Гүйцэтгэсэн: О.Саруулгэрэл /22B1NUM1637/

Шалгасан: Г.Уганбаяр

2025 он

Улаанбаатар хот

ГАРЧИГ

PATHWAY 1: Kotlin fundamentals

Conditionals

→ If/else statement

Syntax:

```
if(condition){
        body1
} else if(condition){
body2
} else {
        body3
}
```

→ When statement

Syntax:

```
When(parameter){
condition1 -> body1
        condition2 -> body2
        condition3, condition4 -> body3&body4
        else -> body5 //optional
}
```

Use the *in* keyword for a range of conditions

Syntax:

```
When(parameter){
in range_start .. range_end -> body1
condition -> body2
}
```

Use the *is* keyword to check data type

Syntax:

```
When(parameter){
is type -> body1
}
```

→ Use if/else and when as expressions

Syntax:

```
var name = if(condition){
                body1
        } else if(condition){
        body2
        } else {
                body3
        }
```

Nullabillity

It refers to the ability of variables to have an absence of value. In Kotlin, nullability is intentionally treated to achieve null safety.

```
fun main() {
    val favoriteActor = null
    println(favoriteActor)
} // null
```

Non-nullable and nullable variables

- Nullable types are variables that *can* hold null. (?)
  Kotlin-д **null** утга авах боломжтой (nullable) эсвэл авах боломжгүй (non-nullable) хувьсагчийг зааглахын тулд ? тэмдэгтийг ашиглана.

  ```
  // Nullable (null утга авч болно)
  var nullableName: String? = "aaa"
  nullableName = null // Зөвшөөрөгдөнө
  ```

- Non-null types are variables that *can't* hold null.
  ```
  // Non-nullable (null утга авах боломжгүй)
  var name: String = "aaa"
  // name = null // Алдаа! "String" төрөл нь null утга авах боломжгүй.
  ```

Access a property of a nullable variable

Nullable хувьсагчийг ашиглахдаа **null** утгаас үүсэх алдаанаас зайлсхийхийн тулд дараах аргуудыг хэрэглэнэ.

- Safe Call (?.)
  Хэрэв хувьсагч null бол ?. оператор нь кодыг ажиллуулахгүй.

  ```
  var text: String? = "Hello"
  println(text?.length) // 5
  text = null
  println(text?.length) // null
  ```

- Elvis Operator (?:)
  Null байвал default утга өгнө

  ```
  val name = nullableVariable?.method/property ?: defaultValue
  ```

  Жишээ нь:
  ```
  var username: String? = null
  println(username?.length ?: "No username") // "No username"
  ```

- Not-null Assertion (!!)
  Хэрэв хувьсагч null биш гэдэгт итгэлтэй байвал !! ашиглаж болно. Хэрэв null байвал NullPointerException үүсгэнэ.

```
fun main() {
    var favoriteActor: String? = "Sandra Oh"
    println(favoriteActor!!.length)
} // 9

fun main() {
    var favoriteActor: String? = null
    println(favoriteActor!!.length)
} // You get a NullPointerException error
```

- If/else conditionals

```
fun main() {
    var favoriteActor: String? = null

    if(favoriteActor != null) {
      println("The number of characters in your favorite actor's name is
${favoriteActor.length}.")
    } else {
      println("You didn't input a name.")
    }
}
```

- If/else expressions

```
fun main() {
    var favoriteActor: String? = "Sandra Oh"

    val lengthOfName = if (favoriteActor != null) {
      favoriteActor.length
    } else {
      0
    }

    println("The number of characters in your favorite actor's name is $lengthOfName.")
}
```

Classes and Objects

Four basic concepts of OOP:

- **Encapsulation** – Өгөгдөл болон функцийг нэг обьектод нэгтгэж, гаднаас шууд хандахыг хязгаарлана. (private, protected)

- **Abstraction** – Зөвхөн гол онцлогийг харуулж, хэрэггүй нарийн деталиудыг нуух. (abstract class, interface)
- **Inheritance** – Нэг классын шинж чанар, үйлдлүүдийг өөр класс дахин ашиглах боломжтой болгох. (extends, implements)
- **Polymorphism** – Нэг функцийг өөр өөр объектууд олон янзаар хэрэгжүүлэх. (method overriding, method overloading)

A class consists of three major parts:

- **Properties.** Variables that specify the attributes of the class's objects.
- **Methods.** Functions that contain the class's behaviors and actions.
- **Constructors.** A special member function that creates instances of the class throughout the program in which it's defined.

```
class SmartDevice{
        class SmartDevice constructor() {
                ...
        }
}
fun main(){
   val smartDevices = SmartDevice()
}
```

Constructors

- **Primary constructor**
  class-ын нэрийн хажууд тодорхойлогддог бөгөөд **property (шинж чанар)**-уудыг шууд тодорхойлох боломжтой.

  ```
  class ClassName(param1: Type1, param2: Type2, ...) { }
  ```

  Kotlin-д **primary constructor** нь init блок дотор анхны утгуудыг оноож, нэмэлт код бичих боломжтой.
  - Объект үүсэх үед автоматаар ажиллана.
  - Нэг класст **олон init блок** байж болно (дарааллаар гүйцэтгэгдэнэ).

  ```
  class Car(val model: String, val year: Int) {
        init {
            println("Car $model ($year) created.")
        }
  }
  val car = Car("Tesla", 2023) // Car Tesla (2023) created.
  ```

- **Secondary constructor**

```
class ClassName {
        constructor(param1: Type1, param2: Type2, ...)
        { // Initialization logic }
}
```

Хэрэв primary constructor байгаа үед secondary constructor ашиглая гэвэл **заавал primary constructor-ыг дуудах** ёстой (this ашиглан).

```
class SmartDevice(val name: String, val category: String) {
        var deviceStatus = "online"


        constructor(name: String, category: String, statusCode:
Int) : this(name, category) {
        deviceStatus = when (statusCode) {
        0 -> "offline"
        1 -> "online"
        else -> "unknown"
        }
        }
         ...
}
```

Inheritance
Kotlin-д **class нь default-аар final байдаг**, өөрөөр хэлбэл **өөр класст удамшуулах (inheritance) болон override хийх боломжгүй**.
Хэрэв class-аа **өргөтгөх (extend) болон өөрчлөх (override) боломжтой** болгохыг хүсвэл **open түлхүүр үг** ашигладаг.

Quiz

1. The following code will print "Divisible by 5" if number is equal to 25.

```
if (number % 10 == 0) {
      println("Divisible by 10")
} else if (number == 5) {
      println("Divisible by 5") }
```

   a. true
   b. false

2. Which of the following conditions are satisfied when x = 5?

   Choose as many answers as you see fit.

   a. x == 5
   b. x in 1..5

c. x is Int

d. x % 5

3. Which is not a basic concept of object-oriented programming
   a. Abstraction
   b. Readability
   c. Inheritance
   d. Polymorphism
4. Which are the four visibility modifiers in Kotlin?
   a. public, private, protected, abstract
   b. static, override, internal, external
   c. private, protected, public, internal
   d. public, protected, static, internal
5. The ___ keyword is used to call a method from the parent class.
   a. this
   b. super
   c. parent
   d. self
6. A(n) ___ defines properties or methods that a class needs to implement.
   a. Delegate
   b. Generic type
   c. Interface
   d. Subclass
7. Which of the following is best represented by a nullable type?
   a. The number of followers (0 or more) in a social media app.
   b. An optional profile picture.
   c. A username that must be at least one character.
   d. A unique ID given to every user.
8. The ___ operator allows you to call a method only if the object is non-null.
   a. .
   b. !!
   c. ?:
   d. ?.
9. Which is not true of functions in Kotlin?
   a. A function can be changed to another data type, and vice versa.
   b. A function can be returned from another function.
   c. A function can take another function as a parameter.
   d. A function has a data type, such as (Int) -> Unit.

10. A function literal is another name for a ___
    a. Function type
    b. Lambda expression
    c. Function reference
    d. Trailing lambda

PATHWAY 2: Add a button to an app

Quiz

1. Use a ___ Composable to display an image
    a. Button
    b. Text
    c. Image
    d. Icon
2. Alignment.Center centers UI components both horizontally and vertically.
    a. True
    b. False
3. Composable functions can store an object in memory using the ___ composable
    a. remember
    b. Column
    c. Modifier
    d. @Composable
4. The debugger allows you to inspect variables when code execution has been suspended.
    a. True
    b. False
5. By using ___ values in a composable function, variables can be made into observables that schedule a recomposition when their value is changed.
    a. remember
    b. Modifier
    c. @Composable
    d. mutableStateOf
6. The ___ composable places its children in a vertical sequence.
    a. Row
    b. Box
    c. Column
    d. Modifier

7. The ___ debugger feature allows you to navigate back up the call stack.
    a. Step over
    b. Step out
    c. Step into
    d. Resume program

PATHWAY 3: Interacting with UI and state

Composition ба Recompostion

Composition гэдэг нь Compose хэрэглүүрийг ажиллуулах үед бүтээгддэг UI-н тодорхойлолт юм. Compose апп нь **composable** функцуудыг ашиглан өгөгдлийг UI болгон хувиргадаг. Хэрэв **state** өөрчлөгдвөл Compose тухайн нөлөөлөгдсөн **composable** функцуудыг дахин ажиллуулж, шинэчилсэн UI-г үүсгэдэг. Үүнийг **recomposition** гэж нэрлэдэг.

Compose нь **recomposition**-ийг автоматаар зохицуулдаг бөгөөд хэрэглэгчийн оролцоо шаардагдахгүй.

State ба MutableState

Compose-д **State** болон **MutableState** төрлүүдийг ашиглан **state**-ийг ажиглагддаг (observable) болгож, UI-г шинэчилдэг.

- **State** – өөрчлөгдөшгүй (immutable), зөвхөн утгыг унших боломжтой.
- **MutableState** – өөрчлөгдөх боломжтой (mutable), утгыг унших болон шинэчлэх боломжтой.

State-г **mutableStateOf()** функц ашиглан үүсгэдэг. Энэ функц анхны утгыг хүлээн авч **State** объектод хадгалж, UI-д өөрчлөлт гарвал **observable** байдлаар дахин зурдаг.

Жишээ код:

```
var count by remember { mutableStateOf(0) }
```

State Hoisting

State hoisting гэдэг нь **state**-ийг **composable** функцээс гадна, илүү хянагдах боломжтой **дээд түвшний компонент** руу шилжүүлэх арга юм. Энэ нь **UI**-г илүү **stateless** болгож, дахин ашиглах боломжийг нэмэгдүүлдэг.

Stateful vs Stateless Composable

- **Stateful Composable** – Дотороо **state** агуулдаг.

- **Stateless Composable** – **state**-ийг өөрөөсөө гадуур хадгалж, зөвхөн өгөгдөл болон event дамжуулах байдлаар ажилладаг.

@StringRes Аннотаци

@StringRes нь **type-safe** аннотаци бөгөөд **string resource** ашиглахад зориулагдсан. Энэ нь **values/strings.xml** файлд хадгалагдсан **нөөц мөр** (string resource)-ийг зөв хэрэглэж буйг шалгах боломжтой болгодог.

```
@Composable
fun Greeting(@StringRes message: Int) {
      Text(stringResource(id = message))
}
```

Quiz

1. Jetpack Compose runs your composables for the first time, during ___ it will keep track of the composables that you call to describe your UI.
   a. Initial composition
   b. Recomposition
   c. State change
   d. App termination
2. The only way to modify a Composition is through recomposition.
   a. True
   b. False
3. ___ is when Jetpack Compose re-executes the composables that may have changed in response to data changes.
   a. Initial composition
   b. Recomposition
   c. State change
   d. App termination
4. ___ in an application is any value that can change over time.
   a. State
   b. value
   c. valueChange
   d. StateValue
5. ___ is a pattern of moving state up to make a component stateless.
   a. State change
   b. State hoisting
   c. Hoist composition

d. Recomposition

6. Which KeyboardAction property is used to move the focus to the next composable?
   a. onDone
   b. onNext
   c. onGo
   d. onSend
7. Which of the following Kotlin functions is used to round up a Double or Float?
   a. kotlin.math.ceilUp()
   b. kotlin.math.ceil()
   c. kotlin.math.roundDown()
   d. kotlin.math.roundUp()
8. Layout Inspector is a tool in Jetpack Compose that allows you to inspect a Compose layout inside a running app in an emulator or physical device.
   a. True
   b. False
9. UI tests are stored in the ___ directory.
   a. main
   b. androidTest
   c. test
   d. res
10. Local tests and UI tests should be annotated with the ___ annotation.
    a. @VisibleForTesting
    b. @Preview
    c. @Test
    d. @Composable