

# Лабораторийн тайлан №3

## Котлин программчлалын хэлэнд төлөвөөр программчлах

МТЭС, МКУТ, Программ хангамж

Г.Батням, 22B1NUM5578

### 1.Ажлын зорилго

Энэ лабораторийн ажлаар товчлуур бүхий апплекэйшин хийж өгөгдсөн даалгавруудыг хийж гүйцэтгэн нийт дасгалуудын бүлгийн асуултуудад хариулна.

### 2.Онолын судалгаа

**Generics** – Өгөгдлийн төрлийг уян хатан байлгах боломжийг олгодог. Өөрөөр хэлбэл, answer-ийн өгөгдлийн төрлийг тодорхойлохдоо тухайн объект үүсгэх үед шийдэх боломжтой гэсэн үг. Ямар нэгэн method гүй байна.

*Жишээ нь:*

```
class Question<T>(  
  
    val questionText: String,  
  
    val answer: T,  
  
    val difficulty: String  
  
)
```

*Тайлбар :* Энд *T* нь **generic placeholder** бөгөөд тухайн объектыг үүсгэх үед **жинхэнэ төрлөөр (String, Boolean, Int гэх мэт)** солигдох болно.

**Enum (Enumeration) class** – хязгаарлагдмал утгуудтай өгөгдлийн төрөл тодорхойлоход ашиглагддаг. Жишээлбэл, дөрвөн зүг (North, South, East, West) эсвэл асуултын хүндрэлтийн түвшин (Easy, Medium, Hard) зэрэг нь enum class-аар илэрхийлэхэд тохиромжтой. Ямар нэгэн method гүй байна.

*Жишээ нь:*

```
enum class Difficulty {  
  
    EASY, MEDIUM, HARD  
  
}
```

**data class** – Зөвхөн өгөгдлийг хадгалах зориулалттай тусгай класс.

**copy()** – Объектыг хуулбарлахдаа ашиглах боломжтой

**toString()** – Тухайн объектын гишүүн өгөгдлийн утгыг string болгоно.

**equals(), hashCode()** – 2 объектыг харицуулах үйлдлийг хийхэд ашиглагдана.

**componentN()** - Өгөгдлийн хэсгүүдийг шууд задлах боломжтой зэрэг методыг автоматаар хэрэгжүүлнэ. new түлхүүр үг хэрэггүй.

*Жишээ нь:*

```
data class Question<T>(  
    val questionText: String,  
    val answer: T,  
    val difficulty: Difficulty  
)
```

**Singleton** – Зөвхөн нэг объект л үүсдэг ба шууд нэрийг нь ашиглаж хандах боломжтой тусгай төрлийн класс юм.

*Жишээ нь:*

```
object StudentProgress {  
    var total: Int = 10  
    var answered: Int = 3  
}
```

**Companion object** – Класс дотор "Singleton" объект байрлуулахын тулд companion object гэсэн түлхүүрийг ашиглана. Ингэснээр эх классын нэрээр шууд тухайн companion object – ийн өгөгдөлд хандах боломжтой.

**Extension Property** – Өргөтгөх шинж чанар класс өргөтгөхдөө, шинээр шинж чанар нэмэхийн тулд **төрлийн нэр + цэг (.) + хувьсагчийн нэр** гэсэн хэлбэртэйгээр тодорхойлдог. Тухайн классыг өргөтгөснөөр шинж чанарыг хадгалах боломжгүй байдаг ба заавал **get()** ашиглах

*Жишээ нь:*

```
val Quiz.StudentProgress.progressText:  
String get() = "${answered} of ${total} answered"
```

**Extension function** – Тухайн классд нэмэлтээр функц нэмж өргөтгөхдөө төрлийн нэр + цэг (.) + **функцийн нэр** гэсэн хэлбэртэйгээр тодорхойлдог.

*Жишээ нь:*

```
fun Quiz.StudentProgress.printProgressBar() {  
}
```

**Interface** – Интерфейс нь тодорхой **шинж чанар (property)**, **функцууд (method)**-ийг заавал хэрэгжүүлэхийг шаарддаг **гэрээ (contract)** юм.

**UpperCamelCase** загварын дагуу нэртэй байх ба дотор нь заавал хэрэгжүүлэх шинж чанар, функцуудыг тодорхойлно. интерфейсээс авсан шинж чанар, функцүүдийг **override** түлхүүр үгээр тодорхойлох ёстой.

**let()** – функц нь объектын нэрийн оронд "it" нэртэй идентификаторыг ашиглан ламбда илэрхийлэлд объектын шинж чанарт хандах боломжийг олгодог.

**apply()** – функц нь объектын шинж чанарыг шууд дууддаг бөгөөд энэ объектын хувьсагчийг үүсгэх шаардлагагүй. **apply()** нь тухайн объектыг эргүүлж буцааж өгдөг бөгөөд энэ нь тухайн объектын аргуудыг дуудах боломжийг олгодог.

*Жишээ нь:*

```
val quiz = Quiz().apply {  
    printQuiz()  
}
```

**List** – уншихад зориулсан зөвхөн унших боломжтой, дараалсан цуглуулгын шинж чанар ба аргуудыг тодорхойлдог интерфейс юм.

*Жишээ нь: val solarSystem = listOf("Mercury", "Venus")*

**MutableList** – List интерфейсийг өргөтгөж, листийн элементийг нэмэх, устгах зэрэг өөрчлөлт хийх аргуудыг тодорхойлдог интерфейс юм.

*Жишээ нь: val solarSystem = mutableListOf("Mercury", "Venus")*

**Set** - Тодорхой дараалалгүй, давхардсан утгуудыг зөвшөөрөхгүй цуглуулга юм. Элементийг хайх нь хурдан ба list – тэй харьцуулахад илүү их санах ой зарцуулдаг.**contains** method нь тухайн тэмдэгт болон бусад өгөгдлийн төрлийн зүйл нийт **set, list, map, array** дотор байгаа эсэхийг шалгаад true эсвэл false гэсэн утгын аль нэгийг буцаадаг.

**Map** – Түлхүүрүүд болон утгуудыг агуулсан цуглуулга юм. Давтагдахгүй түлхүүрүүд нь бусад утгууд руу холбогддог. Түлхүүр ба түүний холбогдсон

утгыг түлхүүр-утгын хослол гэж нэрлэдэг. Map – ийн түлхүүрүүд нь давтагдашгүй харин утгууд нь давхардаж болно.

**forEach()** – функцийг ашиглан List доторх бүх элементүүдийг гүйлгэж болно. lambda дотор it түлхүүр үгийг ашигласнаар тухайн элементэд хандаж болно.

***Ерөнхий загвар : forEach(action: (T) -> Unit)***

***Жишээ нь: cookies.forEach {  
println("Menu item: \${it.name}")  
}***

**map()** – Функц нь нэг Collection-ийг (List, Set, Map гэх мэт) өөр төрлийн Collection болгон хувиргах боломжтой функц юм.

***Жишээ нь:***

***List<Cookie>-ийг List<String> болгон хувиргасан.***

**filter()** нь Collection-оос тодорхой нөхцөлийг хангах элементүүдийг агуулсан шинэ Collection үүсгэдэг. map() функцтэй харьцуулахад filter()-ийн үр дүн нь анхны цуглуулгаас жижигхэн буюу ижил хэмжээтэй байж болно. filter() нь мөн ижил төрлийн Collection буцаадаг.

***Жишээ нь : List<Cookie> дээр filter() ашиглавал үр дүн нь мөн List<Cookie> байна.***

***Мөн манай кодын жишээн дээр it.softBaked шинэ чанар нь true байх тохиолдолд тухайн Cookie-г шинэ жагсаалтанд оруулсан.***

**groupBy()** – List-ийг тодорхой нөхцөлөөр түлхүүр-утга (key-value) бүтэцтэй Map болгон хувиргадаг. Функцэд өгсөн давтагдахгүй утгууд түлхүүрүүд болж, харин анхны List-ийн тухайн түлхүүрт хамаарах бүх элементүүд нь утгуудын жагсаалт болно. Бүлэглэсэн Map-аас тухайн түлхүүрт харгалзах жагсаалтыг [] ашиглан авах боломжтой.

***Жишээ нь:***

***val numbers = listOf(1, 2, 3, 4, 5, 6)  
val groupedNumbers = numbers.groupBy { it % 2 }  
println(groupedNumbers) Гарч ирэх үр дүн: {1=[1, 3, 5], 0=[2, 4, 6]}***

**fold()** – цуглуулгаас (collection) ганц утга гаргахын тулд ашиглагддаг. Энэ нь ихэвчлэн нийт үнэ тооцох, бүх элементүүдийг нийлүүлж дундаж олох гэх мэт тооцоололд хэрэглэгддэг.

***Жишээ нь: Бүх жигнэмэгийн нийт үнийг тооцоолох***

***val totalPrice = cookies.fold(0.0) {  
total, cookie -> total + cookie.price***

}

**Тайлбар:**

*0.0 анхны утга (Double гэж автоматаар тодорхойлогдоно).*

*total нь өмнөх утгыг хадгална.cookie.*

*price нь одоогийн элементээс авах утга.*

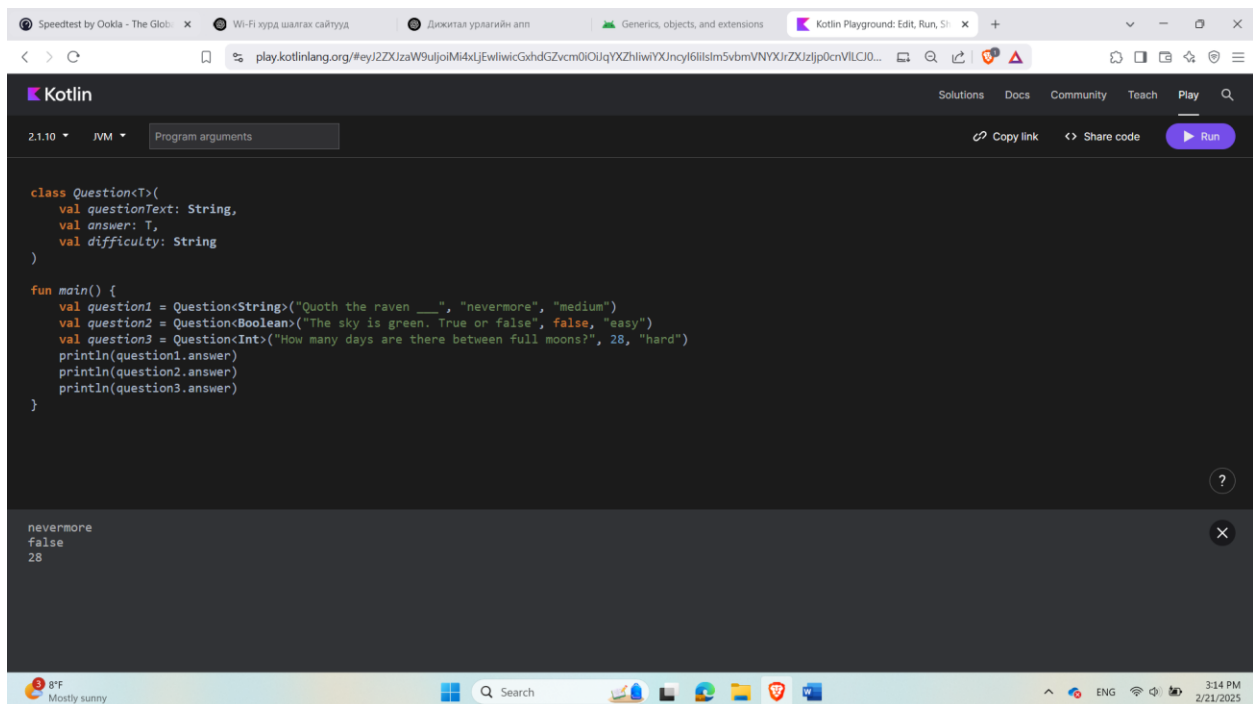
**sort()** – Функци нь энгийн өгөгдлүүд (Int, String гэх мэт)-ийг эрэмбэлж чадна. Харин объектын жагсаалт дээр (List<Cookie>) ямар шинж чанараар эрэмбэлэхээ тодорхойлох шаардлагатай.

**sortedBy()** – ямар шинж чанараар эрэмбэлэхээ lambda-гаар тодорхойлох боломж олгодог.

*Жишээ нь: cookies жагсаалтыг нэрийн дагуу (name) А-Я дарааллаар эрэмбэлнэ.*

*val alphabeticalMenu = cookies.sortedBy { it.name }*

### 3.Хэрэгжүүлэлт

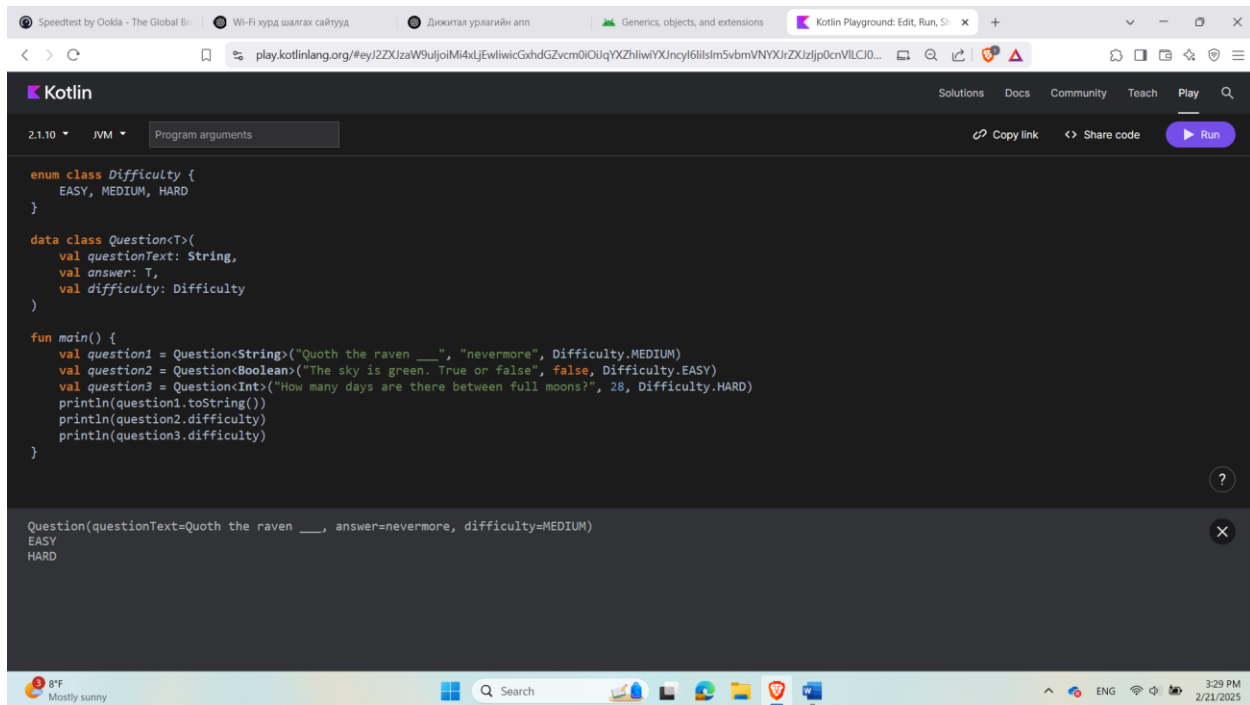


```
class Question<T> {
    val questionText: String,
    val answer: T,
    val difficulty: String
}

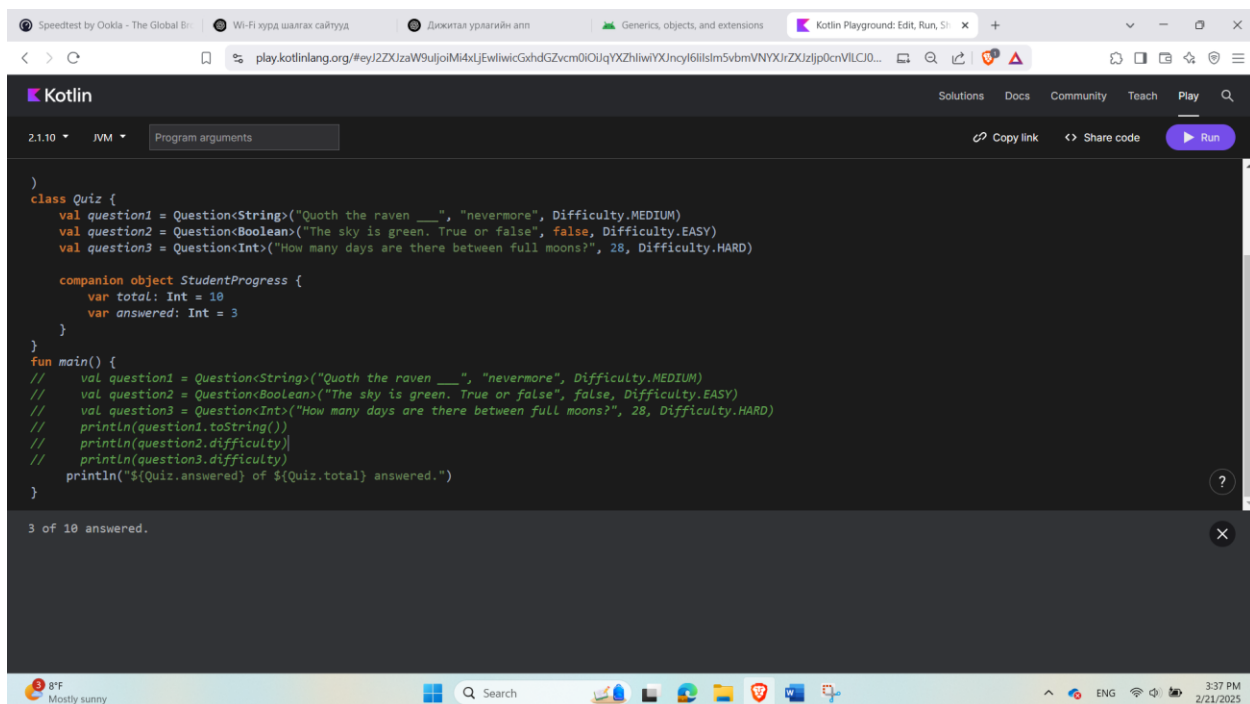
fun main() {
    val question1 = Question<String>("Quoth the raven ____", "nevermore", "medium")
    val question2 = Question<Boolean>("The sky is green. True or false?", false, "easy")
    val question3 = Question<Int>("How many days are there between full moons?", 28, "hard")
    println(question1.answer)
    println(question2.answer)
    println(question3.answer)
}
```

nevermore  
false  
28

*Зураг 1 Generic классыг хэрэгжүүлсэн байдал*



Зураг 2 Data классыг хэрэгжүүлсэн байдал



Зураг 3 Companion болон singleton классыг хэрэгжүүлсэн

The screenshot shows the Kotlin Playground interface. The code defines a `Quiz` class with a `StudentProgress` inner class. The `StudentProgress` class has a `progressText` property and a `printProgressBar` function. The `main` function creates three questions and prints their progress bars. The output shows the progress bars for each question, with the first question being 3 out of 10 answered.

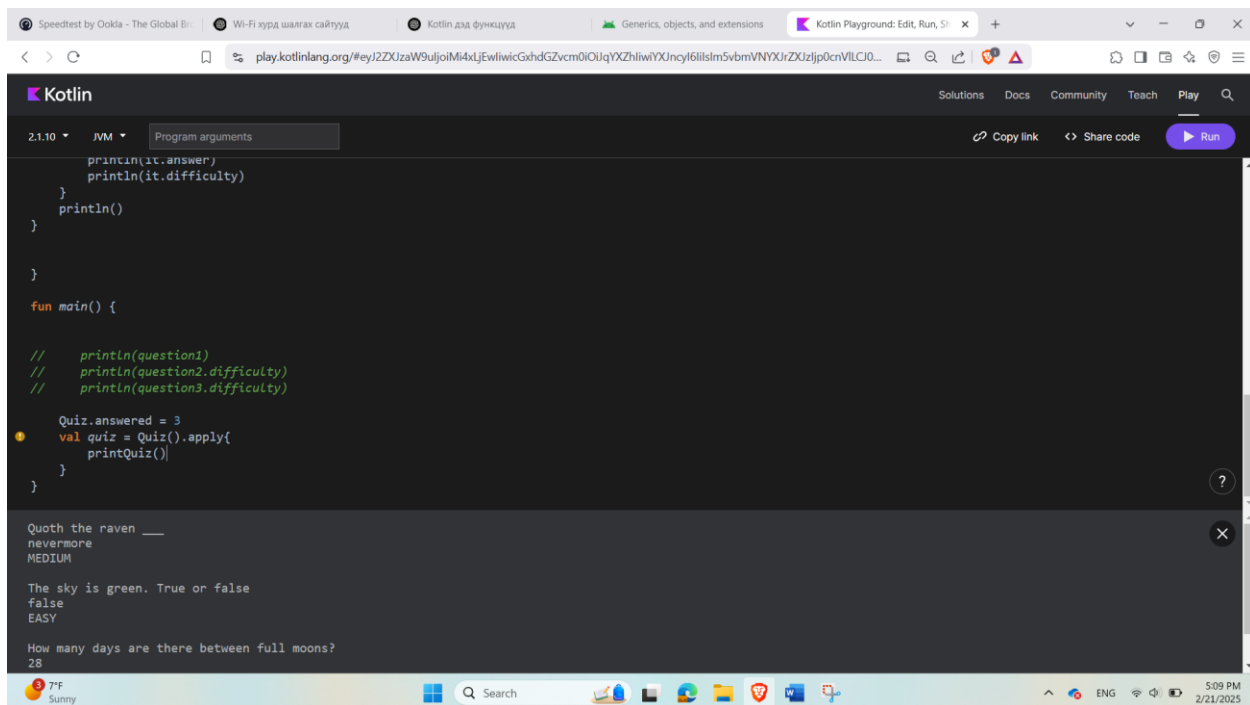
```
3 of 10 answered.
3 of 10 answered
3 of 10 answered
kotlin.Unit
```

Зураг 4 Extension function болон property нэмсэн байдал

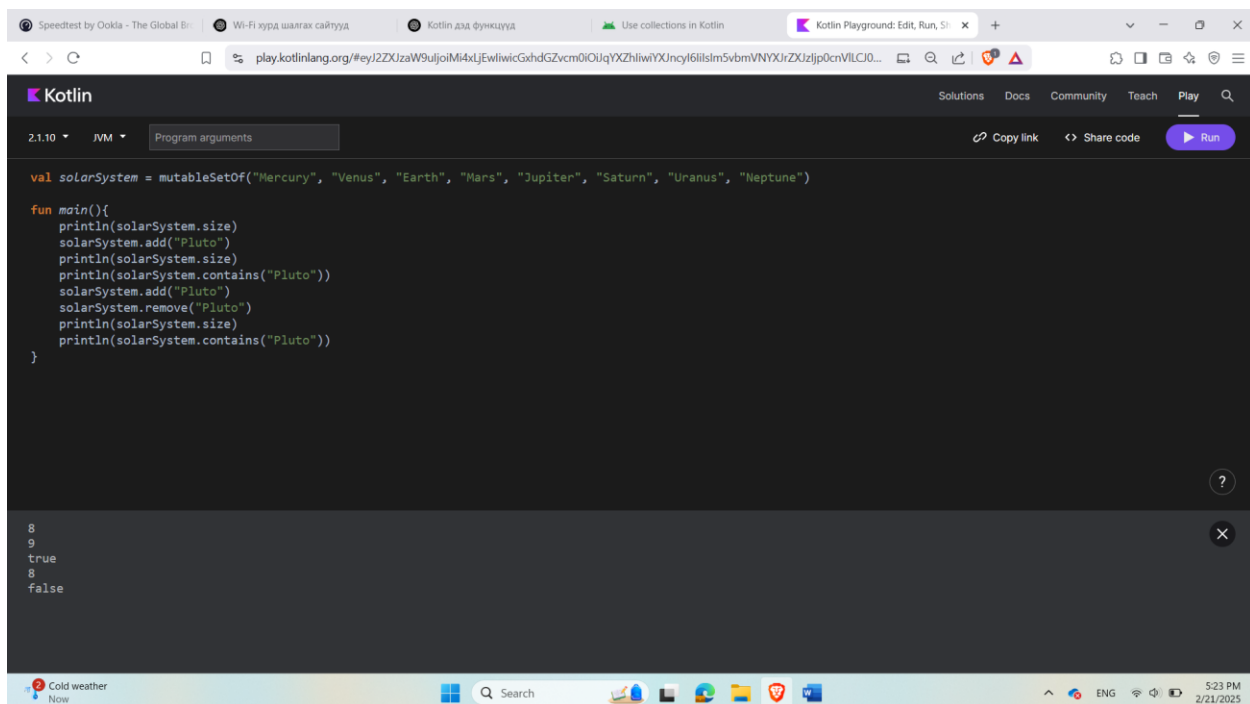
The screenshot shows the Kotlin Playground interface. The code defines an `enum class Difficulty` with values `EASY`, `MEDIUM`, and `HARD`. It also defines an `interface ProgressPrintable` with a `progressText` property and a `printProgressBar` function. The `Quiz` class implements this interface. The `main` function creates a question and prints its progress bar. The output shows the progress bar for the question, with the first question being 3 out of 10 answered.

```
Question(questionText=Quoth the raven ____, answer=nevermore, difficulty=MEDIUM)
EASY
HARD
3 of 10 answered
```

Зураг 5 Interface – г хэрэгжүүлсэн байдал



*Зураг 6 Дэд функцуудыг хэрэгжүүлсэн байдал*



*Зураг 7 Set – г хэрэгжүүлсэн байдал*



The screenshot shows the Kotlin Playground interface. The code defines a `mutableMapOf` for a solar system and a `main` function that prints its size, adds a new planet, and checks for the presence of Mars.

```
val solarSystem = mutableMapOf(
    "Mercury" to 0,
    "Venus" to 0,
    "Earth" to 1,
    "Mars" to 2,
    "Jupiter" to 79,
    "Saturn" to 82,
    "Uranus" to 27,
    "Neptune" to 14
)

fun main(){
    println(solarSystem.size)
    solarSystem["Pluto"] = 5
    println(solarSystem.size)
    println(solarSystem.get("Theia"))
    println(solarSystem.contains("Mars"))
}
```

The output shows the size of the map (8), the size after adding Pluto (9), null for Theia, and true for Mars.

```
8
9
null
true
```

*Зураг 8 Мар цуглуулгыг хэрэгжүүлсэн*

The screenshot shows the Kotlin Playground interface. The code uses `groupBy` to categorize cookies by their `softBaked` status and then prints the names and prices of each group.

```
Cookie("Blueberry Tart", true, true, 1.79),
Cookie("Sugar and Sprinkles", false, false, 1.39)
)

fun main() {
    val groupedMenu = cookies.groupBy { it.softBaked }

    val softBakedMenu = groupedMenu[true] ?: emptyList()
    val crunchyMenu = groupedMenu[false] ?: emptyList()

    println("Soft cookies:")
    softBakedMenu.forEach {
        println("${it.name} - ${it.price}")
    }

    println("Crunchy cookies:")
    crunchyMenu.forEach {
        println("${it.name} - ${it.price}")
    }
}
```

The output lists the prices for soft and crunchy cookies.

```
Soft cookies:
Banana Walnut - $1.49
Snickerdoodle - $1.39
Blueberry Tart - $1.79
Crunchy cookies:
Chocolate Chip - $1.69
Vanilla Creme - $1.59
Chocolate Peanut Butter - $1.49
Sugar and Sprinkles - $1.39
```

*Зураг 9 GroupBy функцийг хэрэгжүүлсэн*

The screenshot shows the Kotlin Playground interface with the following code:

```
Cookie("Blueberry Tart", true, true, 1.79),
Cookie("Sugar and Sprinkles", false, false, 1.39)
)

fun main() {
    val groupedMenu = cookies.groupBy { it.softBaked }

    val softBakedMenu = groupedMenu[true]?: emptyList()
    val crunchyMenu = groupedMenu[false]?: emptyList()
    val totalPrice = cookies.fold(0.0) { total, cookie ->
        total + cookie.price
    }
    println("Total price: $$totalPrice")
    println("Soft cookies:")
    softBakedMenu.forEach {
        println("${it.name} - ${it.price}")
    }

    println("Crunchy cookies:")
    crunchyMenu.forEach {
        println("${it.name} - ${it.price}")
    }
}
```

The output of the program is:

```
Total price: $10.83
Soft cookies:
Banana Walnut - $1.49
Snickerdoodle - $1.39
Blueberry Tart - $1.79
Crunchy cookies:
Chocolate Chip - $1.69
Vanilla Creme - $1.59
Chocolate Peanut Butter - $1.49
Sugar and Sprinkles - $1.39
```

*Зураг 10 Fold() функцийг хэрэгжүүлсэн*

The screenshot shows the Kotlin Playground interface with the following code:

```
fun main() {
    val groupedMenu = cookies.groupBy { it.softBaked }

    val softBakedMenu = groupedMenu[true]?: emptyList()
    val crunchyMenu = groupedMenu[false]?: emptyList()
    val totalPrice = cookies.fold(0.0) { total, cookie ->
        total + cookie.price
    }
    println("Total price: $$totalPrice")
    println("Soft cookies:")
    softBakedMenu.forEach {
        println("${it.name} - ${it.price}")
    }
    val alphabeticalMenu = cookies.sortedBy {
        it.name
    }
    println("Alphabetical menu:")
    alphabeticalMenu.forEach {
        println(it.name)
    }
}
```

The output of the program is:

```
Snickerdoodle - $1.39
Blueberry Tart - $1.79
Alphabetical menu:
Banana Walnut
Blueberry Tart
Chocolate Chip
Chocolate Peanut Butter
Snickerdoodle
Sugar and Sprinkles
Vanilla Creme
Crunchy cookies:
```

*Зураг 11 Sortedby() функцийг хэрэгжүүлсэн*

The screenshot shows the Kotlin Playground interface. The code defines a data class `Event` with properties `title`, `description`, `daypart`, and `duration`. The `main` function creates an `Event` instance and prints it. The output shows the event details.

```
data class Event(
    val title : String,
    val description : String?,
    val daypart : String,
    val duration : Int
)

fun main(){
    val events = Event(
        " Study Kotlin",
        "Commit to studying Kotlin at least 15 minutes per day.",
        " Evening",
        15
    )
    println(events)
}
```

Event(title= Study Kotlin, description=Commit to studying Kotlin at least 15 minutes per day., daypart= Evening, duration=15)

*Зураг 12 1дэх task – ийг амжилттай хэрэгжүүлсэн*

The screenshot shows the Kotlin Playground interface. The code defines an enum class `Daypart` with values `MORNING`, `AFTERNOON`, and `EVENING`. It also defines a data class `Event` with properties `title`, `description`, `daypart`, and `duration`. The `main` function creates an `Event` instance with `Daypart.MORNING` and prints it. The output shows the event details.

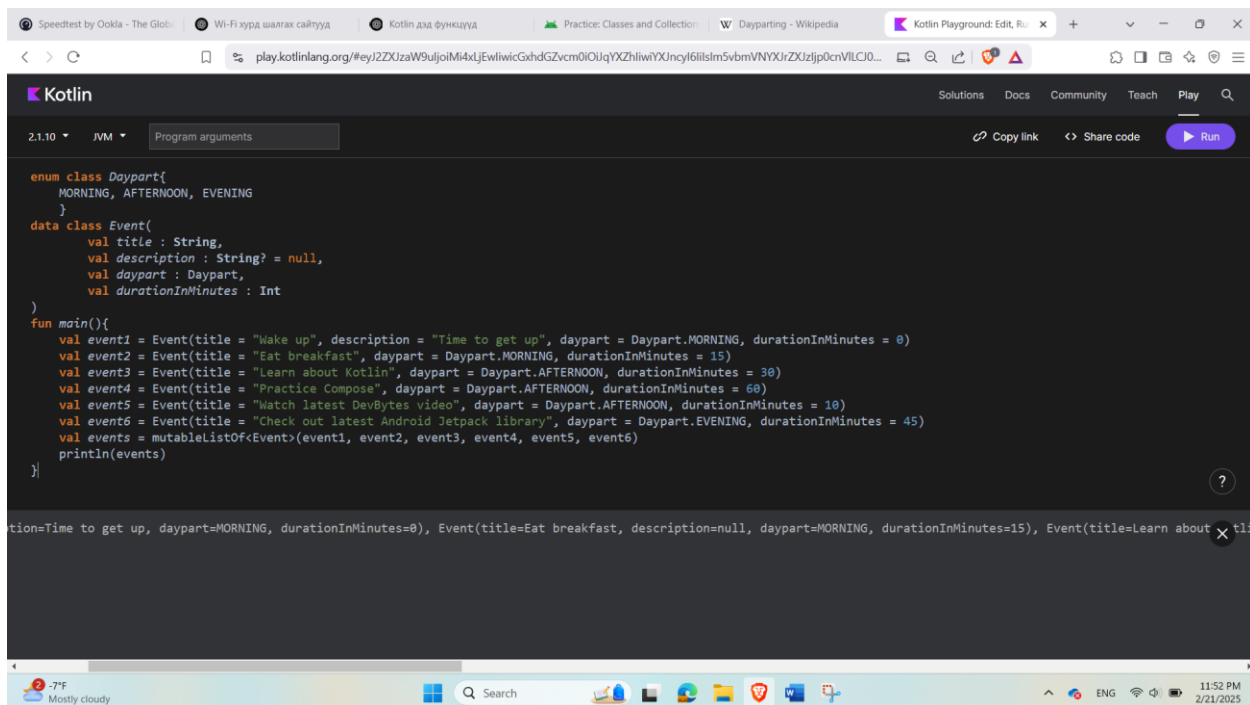
```
enum class Daypart{
    MORNING, AFTERNOON, EVENING
}

data class Event(
    val title : String,
    val description : String?,
    val daypart : Daypart,
    val duration : Int
)

fun main(){
    val events = Event(
        " Study Kotlin",
        "Commit to studying Kotlin at least 15 minutes per day.",
        Daypart.MORNING,
        15
    )
    println(events)
}
```

Event(title= Study Kotlin, description=Commit to studying Kotlin at least 15 minutes per day., daypart=MORNING, duration=15)

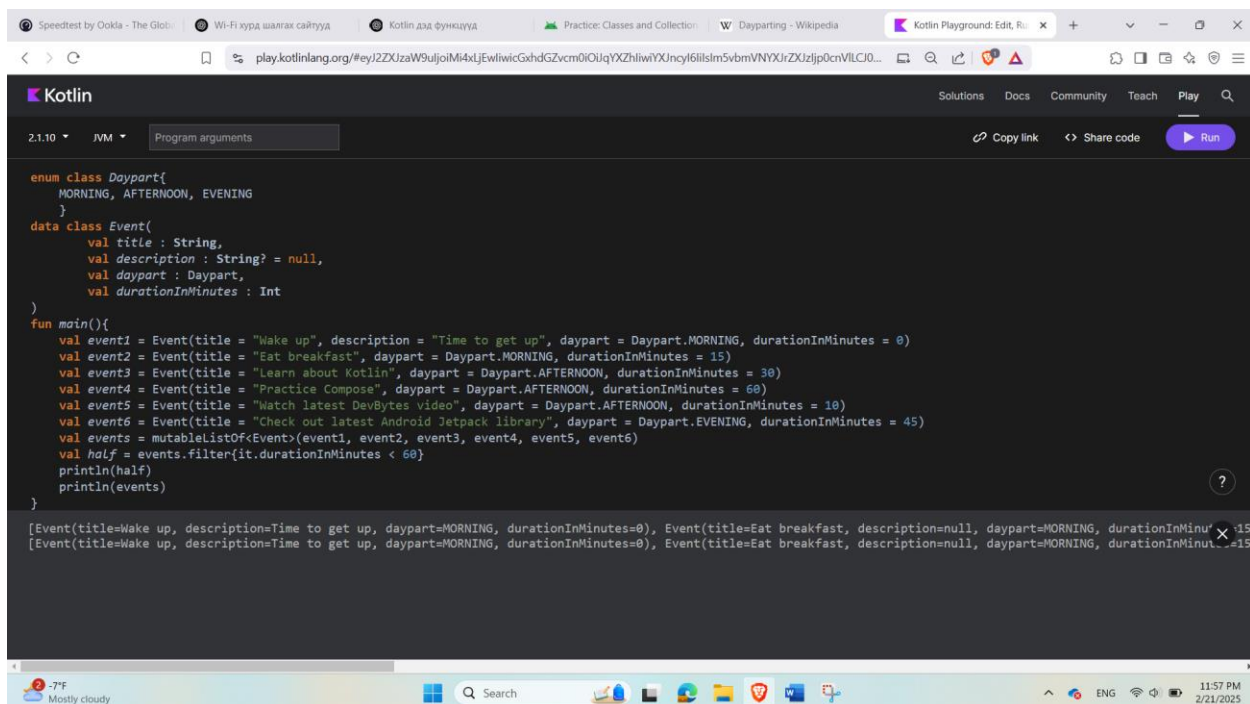
*Зураг 13 2дах task – ийг амжилттай хэрэгжүүлсэн*



```
enum class Daypart{
    MORNING, AFTERNOON, EVENING
}
data class Event(
    val title : String,
    val description : String? = null,
    val daypart : Daypart,
    val durationInMinutes : Int
)
fun main(){
    val event1 = Event(title = "Wake up", description = "Time to get up", daypart = Daypart.MORNING, durationInMinutes = 0)
    val event2 = Event(title = "Eat breakfast", daypart = Daypart.MORNING, durationInMinutes = 15)
    val event3 = Event(title = "Learn about Kotlin", daypart = Daypart.AFTERNOON, durationInMinutes = 30)
    val event4 = Event(title = "Practice Compose", daypart = Daypart.AFTERNOON, durationInMinutes = 60)
    val event5 = Event(title = "Watch latest DevBytes video", daypart = Daypart.AFTERNOON, durationInMinutes = 10)
    val event6 = Event(title = "Check out latest Android Jetpack library", daypart = Daypart.EVENING, durationInMinutes = 45)
    val events = mutableListOf<Event>(event1, event2, event3, event4, event5, event6)
    println(events)
}
```

tion=Time to get up, daypart=MORNING, durationInMinutes=0), Event(title=Eat breakfast, description=null, daypart=MORNING, durationInMinutes=15), Event(title=Learn about Kotlin, description=Learn about Kotlin, daypart=AFTERNOON, durationInMinutes=30), Event(title=Practice Compose, description=Practice Compose, daypart=AFTERNOON, durationInMinutes=60), Event(title=Watch latest DevBytes video, description=Watch latest DevBytes video, daypart=AFTERNOON, durationInMinutes=10), Event(title=Check out latest Android Jetpack library, description=Check out latest Android Jetpack library, daypart=EVENING, durationInMinutes=45)

*Зураг 14 3дэх task – ийг амжилттай хэрэгжүүлсэн*



```
enum class Daypart{
    MORNING, AFTERNOON, EVENING
}
data class Event(
    val title : String,
    val description : String? = null,
    val daypart : Daypart,
    val durationInMinutes : Int
)
fun main(){
    val event1 = Event(title = "Wake up", description = "Time to get up", daypart = Daypart.MORNING, durationInMinutes = 0)
    val event2 = Event(title = "Eat breakfast", daypart = Daypart.MORNING, durationInMinutes = 15)
    val event3 = Event(title = "Learn about Kotlin", daypart = Daypart.AFTERNOON, durationInMinutes = 30)
    val event4 = Event(title = "Practice Compose", daypart = Daypart.AFTERNOON, durationInMinutes = 60)
    val event5 = Event(title = "Watch latest DevBytes video", daypart = Daypart.AFTERNOON, durationInMinutes = 10)
    val event6 = Event(title = "Check out latest Android Jetpack library", daypart = Daypart.EVENING, durationInMinutes = 45)
    val events = mutableListOf<Event>(event1, event2, event3, event4, event5, event6)
    val half = events.filter{it.durationInMinutes < 60}
    println(half)
    println(events)
}
```

[Event(title=Wake up, description=Time to get up, daypart=MORNING, durationInMinutes=0), Event(title=Eat breakfast, description=null, daypart=MORNING, durationInMinutes=15), Event(title=Learn about Kotlin, description=Learn about Kotlin, daypart=AFTERNOON, durationInMinutes=30), Event(title=Practice Compose, description=Practice Compose, daypart=AFTERNOON, durationInMinutes=60), Event(title=Watch latest DevBytes video, description=Watch latest DevBytes video, daypart=AFTERNOON, durationInMinutes=10), Event(title=Check out latest Android Jetpack library, description=Check out latest Android Jetpack library, daypart=EVENING, durationInMinutes=45)]

*Зураг 15 4дэх task – ийг амжилттай хийж гүйцэтгэсэн*

The screenshot shows the Kotlin Playground interface with the following code:

```
val title : String,
val description : String? = null,
val daypart : Daypart,
val durationInMinutes : Int
)
fun main(){
    val event1 = Event(title = "Wake up", description = "Time to get up", daypart = Daypart.MORNING, durationInMinutes = 0)
    val event2 = Event(title = "Eat breakfast", daypart = Daypart.MORNING, durationInMinutes = 15)
    val event3 = Event(title = "Learn about Kotlin", daypart = Daypart.AFTERNOON, durationInMinutes = 30)
    val event4 = Event(title = "Practice Compose", daypart = Daypart.AFTERNOON, durationInMinutes = 60)
    val event5 = Event(title = "Watch latest DevBytes video", daypart = Daypart.AFTERNOON, durationInMinutes = 10)
    val event6 = Event(title = "Check out latest Android Jetpack library", daypart = Daypart.EVENING, durationInMinutes = 45)
    val events = mutableListOf<Event>(event1, event2, event3, event4, event5, event6)
    val half = events.filter{it.durationInMinutes < 60}
    val buleg = events.groupBy {it.daypart}
    buleg.forEach { (daypart, events) ->
        println("$daypart: ${events.size} events")
    }
    println(half)
    println(events)
}
```

The output shows the number of events for each daypart and the filtered list of events with duration less than 60 minutes:

```
MORNING: 2 events
AFTERNOON: 3 events
EVENING: 1 events
[Event(title=Wake up, description=Time to get up, daypart=MORNING, durationInMinutes=0), Event(title=Eat breakfast, description=null, daypart=MORNING, durationInMinutes=15)
[Event(title=Wake up, description=Time to get up, daypart=MORNING, durationInMinutes=0), Event(title=Eat breakfast, description=null, daypart=MORNING, durationInMinutes=15)
```

*Зураг 16 5дэх task – ийг амжилттай хийж гүйцэтгэсэн*

The screenshot shows the Kotlin Playground interface with the following code:

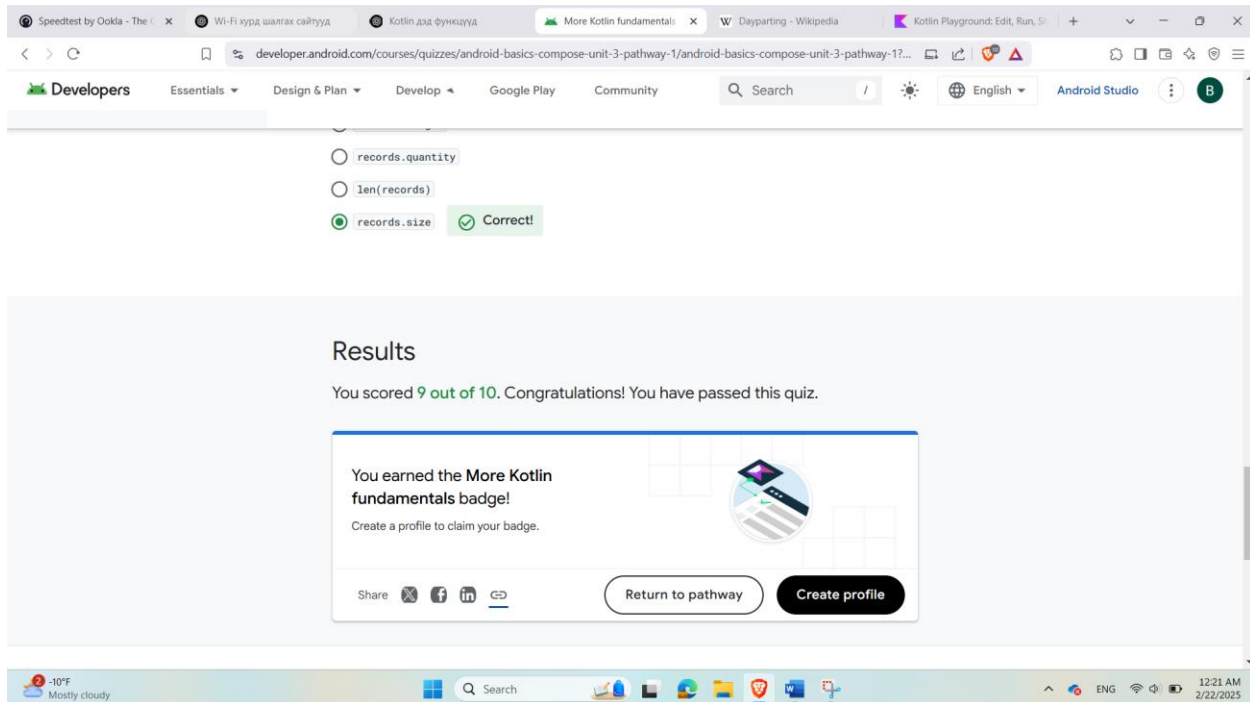
```
val description : String? = null,
val daypart : Daypart,
val durationInMinutes : Int
)
fun main(){
    val event1 = Event(title = "Wake up", description = "Time to get up", daypart = Daypart.MORNING, durationInMinutes = 0)
    val event2 = Event(title = "Eat breakfast", daypart = Daypart.MORNING, durationInMinutes = 15)
    val event3 = Event(title = "Learn about Kotlin", daypart = Daypart.AFTERNOON, durationInMinutes = 30)
    val event4 = Event(title = "Practice Compose", daypart = Daypart.AFTERNOON, durationInMinutes = 60)
    val event5 = Event(title = "Watch latest DevBytes video", daypart = Daypart.AFTERNOON, durationInMinutes = 10)
    val event6 = Event(title = "Check out latest Android Jetpack library", daypart = Daypart.EVENING, durationInMinutes = 45)
    val events = mutableListOf<Event>(event1, event2, event3, event4, event5, event6)
    val half = events.filter{it.durationInMinutes < 60}
    val buleg = events.groupBy {it.daypart}
    buleg.forEach { (daypart, events) ->
        println("$daypart: ${events.size} events")
    }
    println("Last event of the day: ${events.last().title}")
    println(half)
    println(events)
}
```

The output shows the number of events for each daypart, the title of the last event of the day, and the filtered list of events with duration less than 60 minutes:

```
MORNING: 2 events
AFTERNOON: 3 events
EVENING: 1 events
Last event of the day: Check out latest Android Jetpack library
[Event(title=Wake up, description=Time to get up, daypart=MORNING, durationInMinutes=0), Event(title=Eat breakfast, description=null, daypart=MORNING, durationInMinutes=15)
[Event(title=Wake up, description=Time to get up, daypart=MORNING, durationInMinutes=0), Event(title=Eat breakfast, description=null, daypart=MORNING, durationInMinutes=15)
```

*Зураг 17 6дах task – ийг амжилттай хийж гүйцэтгэсэн*

## 4. Бүлгийн асуулт



*Зураг 1 Эхний бүлгийн асуултуудад хариулсан*

## 5. Дүгнэлт

Энэ лабораторийн ажлаар товчлуур бүхий апп, тухайн апп – д list үүсгэн мөн animations хэрэгжүүллээ.

## 6. Ашигласан материал

(<https://developer.android.com/courses/pathways/android-basics-compose-unit-3-pathway-1>, n.d.)

([https://developer.android.com/courses/pathways/android-basics-compose-unit-3-pathway-2?\\_gl=1\\*4nx32f\\*\\_up\\*MQ...](https://developer.android.com/courses/pathways/android-basics-compose-unit-3-pathway-2?_gl=1*4nx32f*_up*MQ...), n.d.)

(<https://developer.android.com/courses/pathways/android-basics-compose-unit-3-pathway-3>, n.d.)