
# Лабораторийн ажил 5

## Unit 5: Connect to the internet

Гүйцэтгэсэн: О.Саруулгэрэл /22B1NUM1637/

Шалгасан: Г.Ууганбаяр

2025 он

Улаанбаатар хот

# PATHWAY 1: GET DATA FROM THE INTERNET

## Coroutine

- **Тодорхойлолт**: Корутин нь "cooperative routines" буюу хамтран ажилладаг дэд программууд юм. Энэ нь гүйцэтгэлийг түр зогсоож (suspend), дараа нь сэргээж (resume) ажиллуулдаг онцлогтой.
- **Онолын суурь**:
    - Корутин нь уламжлалт thread-ээс хөнгөн бөгөөд олон тооны корутиныг нэг thread дээр ажиллуулж болно. Энэ нь CPU-ийн нөөцийг хэмнэж, context switching-ийн зардлыг бууруулдаг.
    - Корутин нь **event loop**-ийн тусламжтайгаар ажилладаг бөгөөд suspend болоход thread-ийг чөлөөлж, бусад ажлыг гүйцэтгэх боломж өгдөг.

### Suspend функц

suspend функц нь гүйцэтгэлийг тодорхой хугацаанд эсвэл нөхцөлд түр зогсоож (pause) боломжтой болгодог. Гэхдээ энэ нь зөвхөн корутин эсвэл өөр suspend функц дотор ажилладаг.

**Хэрхэн зогсдог вэ?**:

- suspend функц дотор **suspension point** (жишээ нь, delay(), await()) байвал тэр цэгт гүйцэтгэл зогсоно.
- Зогссон үед thread чөлөөлөгддөг ба бусад ажлыг гүйцэтгэх боломжтой болно (асинхрон шинжтэй).
- Зогсолт нь яг тухайн suspension point-ийн нөхцлөөс хамаарна (жишээ: delay(1000) бол 1 секунд).

  **Жишээ**:

```
import kotlinx.coroutines.*

fun main() {
    runBlocking {
        println("Weather forecast")
        printForecast() // Suspend функц дуудагдаж байна
        printTemperature()
    }
}

suspend fun printForecast() {
    delay(1000) // Suspension point
    println("Sunny")
}

suspend fun printTemperature() {
    delay(1000) // Suspension point
    println("30\u00b0C")
}
```

  Энд printForecast() болон printTemperature() нь тус бүр 1 секундын delay() агуулсан suspend функцууд юм. Гүйцэтгэл дарааллаар явагдана:

- "Weather forecast" хэвлэгдэнэ.
- printForecast()-д 1 секунд зогсоно, дараа нь "Sunny" хэвлэнэ.
- printTemperature()-д 1 секунд зогсоно, дараа нь "30°C" хэвлэнэ. Нийт хугацаа ~2 секунд болно.

## RunBlocking

runBlocking нь Kotlin Coroutines-д ашиглагддаг бөгөөд асинхрон кодыг блоклох замаар ажиллуулдаг coroutine builder юм. Энэ нь main thread дээр ажиллаж, доторх coroutine-ууд дуусах хүртэл main thread-ийг хүлээлгэж (block) байдаг.

```
import kotlinx.coroutines.*

fun main() {

   runBlocking {

      println("Weather forecast")

      delay(1000) // 1 секундын түр зогсолт

      println("Sunny")

   }

}
```

Энд runBlocking нь ажил дуустал хүлээж, синхрон гүйцэтгэлийг хадгална.

## Launch функц

launch нь Kotlin Coroutines-ийн нэг чухал функц бөгөөд асинхрон ажиллагааг эхлүүлэхэд ашиглагддаг. Энэ нь шинэ coroutine үүсгэж, хамааралгүйгээр (non-blocking) ажиллуулдаг. юу ч буцаадаггүй

```
import kotlinx.coroutines.*

fun main() {
    runBlocking {
        println("Weather forecast")
        launch {
            printForecast()
        }
        launch {
            printTemperature()
        }
    }
}

suspend fun printForecast() {
    delay(1000)
    println("Sunny")
}

suspend fun printTemperature() {
    delay(1000)
    println("30\u00b0C")
}
```

printForecast(), printTemperature() функц тус тусдаа coroutine болсон болохоор илүү хурдан ажиллана().

### Async функц

Кодын асинхрон ажиллагааг илүү үр дүнтэй хийхийн тулд Kotlin Coroutines-д async() ашигладаг. Энэ нь coroutine эхлүүлж, утга (return value) буцаах боломжтой Deferred объект үүсгэдэг.

```
import kotlinx.coroutines.*

fun main() {
    runBlocking {
        println("Weather forecast")
        val forecast: Deferred<String> = async { getForecast() }
        val temperature: Deferred<String> = async { getTemperature() }
        println("${forecast.await()} ${temperature.await()}")
        println("Have a good day!")
    }
}

suspend fun getForecast(): String {
    delay(1000)
    return "Sunny"
}

suspend fun getTemperature(): String {
    delay(1000)
    return "30°C"
}
```

## Exceptions and cancellation

```
suspend fun getTemperature(): String {

    delay(500)

    throw AssertionError("Temperature is invalid") // Алдаа үүсгэж байна

    return "30°C"

}
```

Энэ нь алдаа үүсгэж, coroutine цуцлагдаж (cancelled), програм шууд зогсоно.Бүх coroutine-ууд устаж, parent дотор байгаа getForecast()-ийн үр дүн ч ашиглагдахгүй.

### Exception Handling

try-catch ашигласнаар coroutine-д гарсан алдааг барьж, зохицуулж болно. try-catch ашигласнаар програм шууд зогсохгүй үргэлжлүүлдэг ба алдааг барьж авна.

```
import kotlinx.coroutines.*

fun main() {
    runBlocking {
```

```
        println("Weather forecast")
        try {
            println(getWeatherReport())
        } catch (e: AssertionError) {
            println("Caught exception: $e")
            println("Report unavailable at this time")
        }
        println("Have a good day!")
    }
}
Гаралт нь:
Weather forecast
Caught exception: java.lang.AssertionError: Temperature is invalid
Report unavailable at this time
Have a good day!
```

Хэрэв getTemperature() ажиллахгүй байсан ч, getForecast() үргэлжлэн ажиллаж, ямар нэг үр дүн харуулахыг хүсвэл алдааг async() дотор барьж авах боломжтой.

```
suspend fun getWeatherReport() = coroutineScope {
    val forecast = async { getForecast() }
    val temperature = async {
        try {
            getTemperature()
        } catch (e: AssertionError) {
            println("Caught exception: $e")
            "{ No temperature found }"
        }
    }
    "${forecast.await()} ${temperature.await()}"
}
Гаралт нь:
Weather forecast
Caught exception: java.lang.AssertionError: Temperature is invalid
Sunny { No temperature found }
Have a good day!
```

## Cancellation

Корутинүүдийг цуцлах боломжтой бөгөөд энэ нь хэрэглэгчийн хүсэлт, сүлжээний саатал эсвэл аппын нөөцийг зөв удирдахад ашиглагддаг.

```
suspend fun getWeatherReport() = coroutineScope {
    val forecast = async { getForecast() }
    val temperature = async { getTemperature() }

    delay(200) // 200 мс хүлээгээд
    temperature.cancel() // Температур авах coroutine-г зогсооно

    "${forecast.await()}"
}
```

Temperature  л цуцлагдана. Бусад coroutine-үүдийг нөлөөлөхгүй.

## Coroutine Concepts

Coroutines нь **Structured Concurrency** зарчмыг баримталдаг бөгөөд үүнийг хэрэгжүүлэхийн тулд хэд хэдэн чухал ойлголтууд байдаг:

### Job

Котлин Coroutine-д `launch()` нь Job объект буцаадаг. Job нь coroutine-ийн төлөвийг удирдаж, цуцлах, шалгах боломжийг олгоно.

```
val job = launch {
    ...

    val childJob = launch { ... }

    ...
}
```

#### Job Hierarchy

Котлин Coroutines нь эцэг-хүүхэд (parent-child) харилцаатай ажилладаг. Нэг coroutine өөр coroutine үүсгэвэл, тэдгээр нь шаталсан харилцаатай болно. Үүнийг Job Hierarchy (Ажил үүргийн шатлал) гэж нэрлэдэг.

- Parent цуцлагдвал, бүх Child coroutine-ууд цуцлагдана
- Хүүхэд coroutine цуцлагдвал, эцэг нь нөлөөлөхгүй
- Хэрэв нэг child coroutine алдаа гаргавал, эцэг болон бусад хүүхэд coroutine-ууд цуцлагдана

### CoroutineScope

“coroutineScope() нь coroutine-уудыг дотооддоо зэрэг ажиллуулдаг гэхдээ функц нь бүх coroutine дууссаны дараа л үргэлжилдэг. Энэ нь асинхрон ажиллагаатай боловч синхрон мэт харагдах боломжийг олгодог.”

CoroutineScope нь coroutine-уудыг тодорхой хүрээнд удирдаж, lifecycle-тэй холбох үүрэгтэй. CoroutineScopeашигласнаар нэг coroutine дуусах үед түүний бүх хүүхэд coroutine-ууд автоматаар цуцлагдах бөгөөд удирдаагүй coroutine-ууд үүсэхээс сэргийлнэ. Жишээ нь, хэрэв  Activity дотор lifecycleScope ашиглан coroutine эхлүүлсэн бол, Activity устах үед lifecycleScope нь автоматаар цуцлагдаж, түүний бүх хүүхэд coroutine-ууд мөн цуцлагдана.

#### CoroutineContext

CoroutineContext нь coroutine хэрхэн, хаана ажиллахыг тодорхойлдог мэдээллийг хадгалдаг контекст (environment) юм.

CoroutineContext нь key-value бүтэцтэй бөгөөд дараах үндсэн элементүүдийг агуулдаг:

1. name – Coroutine-ийн нэр, ялгах тэмдэг.
2. job – Coroutine-ийн lifecycle-ийг удирдах.
3. dispatcher – Coroutine-г ямар thread дээр ажиллуулахыг тодорхойлох.

4. exception handler – Coroutine доторх алдааг барьж, зохицуулах.

Хэрэв нэг coroutine дотор нөгөө coroutine үүсгэвэл, шинэ coroutine нь эцэг coroutine-ийн CoroutineContext-ийг автоматаар өвлөнө. Гэхдээ шаардлагатай бол тухайн контекстийг override хийж болно.

```
import kotlinx.coroutines.*

fun main() = runBlocking {
    val parentContext = CoroutineName("ParentCoroutine")

    launch(parentContext) {
        println("Running in: ${coroutineContext[CoroutineName]}")

        launch(CoroutineName("ChildCoroutine")) {
            println("Running in: ${coroutineContext[CoroutineName]}")
        }
    }
}
Гаралт:
Running in: ParentCoroutine
Running in: ChildCoroutine
```

**Dispatcher**

Dispatcher нь coroutine-г ямар thread дээр ажиллуулахыг удирддаг CoroutineContext-ийн нэг чухал бүрэлдэхүүн хэсэг юм.

- Dispatchers.Main -> UI Thread дээр ажиллах. UI шинэчлэхэд ашиглана.

  import kotlinx.coroutines.*

  ```
  fun main() = runBlocking {
      launch(Dispatchers.Main) {
          println("Running on UI Thread: ${Thread.currentThread().name}")
      }
  }
  ```

  Android апп дээр UI шинэчлэхэд ашиглагдана.
  Main thread дээр ажиллах тул UI блоклохгүй.
- Dispatchers.IO -> I/O үйлдэл (сүлжээ, файлын уншилт/бичилт) хийхэд тохиромжтой.

  ```
  import kotlinx.coroutines.*

  fun main() = runBlocking {
      launch(Dispatchers.IO) {
          println("Running on IO Thread: ${Thread.currentThread().name}")
      }
  }
  ```

  Сүлжээний хүсэлт, өгөгдөл хадгалах зэрэг үйлдэлд тохиромжтой.
  Үндсэн Thread-ийг блоклохгүй.

- Dispatchers.Default -> CPU-нд их ачаалал өгдөг тооцоолол хийхэд тохиромжтой.

  ```
  import kotlinx.coroutines.*
  ```

```
fun main() = runBlocking {
    launch(Dispatchers.Default) {
        println("Running on Default Thread: ${Thread.currentThread().name}")
    }
}
```

Олон тооны өгөгдөл боловсруулах, зураг боловсруулах, AI тооцоолол хийхэд тохиромжтой.

CPU-г оновчтой ашиглах тул хурдан ажиллана.

## Web Services ба Retrofit

**Retrofit** нь RESTful веб сервисээс өгөгдөл татахад зориулсан гуравдагч талын номын сан юм. Энэ нь:

- **Автоматжуулалт**: Сервертэй харилцах кодыг үүсгэнэ.
- **Арын Thread**: Хүсэлтийг UI thread-ээс тусад нь гүйцэтгэнэ (гацахгүй).
- **Converter**: JSON хариуг String эсвэл бусад объектоор буцаана.
- **Retrofit-ийн давуу тал**:
    - Код бичих хугацааг хэмнэнэ.
    - XML, JSON гэх мэт түгээмэл форматыг дэмждэг.
    - Алдаа багатай, найдвартай.

## Quiz

1. With concurrent programming, code might execute in an order different from how it was written.

    a. True
    b. False

2. Fill-in-the-blanks

    The main thread is responsible for displaying the user interface responding to user input.

3. Which of the following statements are true about coroutine contexts?
*Choose as many answers as you see fit.*

    a. Dispatchers.Default is the best choice for long running tasks involving reading and writing large amounts of data.
    b. Dispatchers.Main can be used for updating the UI but not for long-running tasks.
    c. A Job controls the lifecycle of a coroutine.
    d. Dispatchers.IO is optimized for network I/O, among other background tasks.

4. launch() and async() are extension functions of a ___, which keeps track of any coroutines it creates.

    a. CoroutineScope
    b. Job
    c. Dispatcher
    d. CoroutineContext

5. Which of the following statements are true about structured concurrency and its best practices?
*Choose as many answers as you see fit.*

    a. If a coroutine is canceled, child coroutines should also be canceled.
    b. A parent scope can complete before one or more of its children are completed.
    c. A failure should propagate downward without canceling the parent coroutine.
    d. Coroutines must be launched from a coroutine scope.

6. Which of the following statements are true about web services?
*Choose as many answers as you see fit.*

    a. GET, POST, and DELETE are all examples of HTTP operations.
    b. A URL is a type of URI but not all URIs are URLs.
    c. RESTful services always provide a formatted XML response.
    d. Retrofit is a third-party library for handling JSON from a web service.

7. Retrofit is a third-party library that enables your app to make requests to a(n) ___ web service.

    a. XML
    b. Socket
    c. RESTful
    d. JSON

8. One recommended way to perform a Retrofit network request is with a coroutine launched in the viewModelScope.

    a. True
    b. False

9. To enable your app to make connections to the Internet, add the 'android.permission.INTERNET' permission in the ___ file.

    a. MainActivity
    b. build.gradle
    c. Android manifest
    d. ViewModel

10. The process of turning a JSON result into usable data, as is done with Gson, is called JSON ___.

    a. Serialization

b. Encoding
c. Converting
d. Parsing

# PATHWAY 2: Load and display images from the internet

## Dependency Injection (DI) гэж юу вэ?

Dependency Injection гэдэг нь классын хэрэгцээтэй объектуудыг (dependencies) тухайн класс дотроо үүсгэхгүй, харин гаднаас дамжуулж өгөх арга юм. Ингэснээр код илүү уян хатан, тест хийхэд хялбар болдог.

## Quiz

1. Which of the following is not a common HTTP operation/method:

    a. GET
    b. POST
    c. DELETE
    d. SET

2. The response from a REST web service is commonly formatted in one of the common data transfer formats like XML or JSON.

    a. True
    b. False

3. Which of the following is not true for the Retrofit library:

    a. It is a client library.
    b. It enables your app to make requests to a REST web service.
    c. It converts Kotlin objects to JSON objects.
    d. It is a third-party library.

4. Which of the following applies to a Singleton pattern:

    a. object declarations are used to declare singleton objects in Kotlin.
    b. Ensures that one, and only one, instance of an object is created
    c. Has one global point of access to that object.
    d. All of the above

5. Each JSON object contains the following:

    a. A set of key-value pairs separated by a colon.
    b. A set of key-value pairs separated by a comma.
    c. A set of key-value pairs separated by a semi colon.
    d. None of the above

6. Following Android's recommended app architecture guidelines, an app should have which of the following:

    a. A UI Layer
    b. A Domain Layer
    c. A Data Layer
    d. A Business Layer

7. The advantages of using Dependency Injection (DI) in your app include which of the following:
   *Choose as many answers as you see fit.*

    a. Helps with the reusability of code
    b. Makes refactoring easier
    c. Helps with testing
    d. Makes your app run faster

8. If your app has more than one type of data source, they should all be stored in the same repository for ease of use.

    a. True
    b. False

9. Which of the following is used to replace the Main dispatcher with a TestDispatcher in a local unit test:

    a. runTest
    b. runBlocking
    c. Distpatchers.resetMain()
    d. Dispatchers.setMain()

10. The runTest() function can be used to test suspend functions.

    a. True
    b. False