OTUS

Android Developer

Coroutines

otus.ru

Проверить, идет ли запись

```
if (видно && слышно) {
    chat.print(“+”)
}
```

Ставим “+”, если все хорошо
“-”, если есть проблемы

Тема вебинара

StateMachine
Диаграммы состояния

Николай Кочетков

Руководитель Андроид разработки FlyXO

Об опыте (например):

22 года в IT, 7 лет - играющий тренер Андроид

Телефон / эл. почта / соц. сети:
LinkedIn: https://www.linkedin.com/in/motorro/
GitHub: https://github.com/motorro/
Medium: https://medium.com/@motorro

Правила вебинара

Активно
участвуем

Off-topic обсуждаем
в Slack #**канал группы**
или #general

Задаем вопрос
в чат или голосом

Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое
на активность

Пишем в чат

Говорим голосом

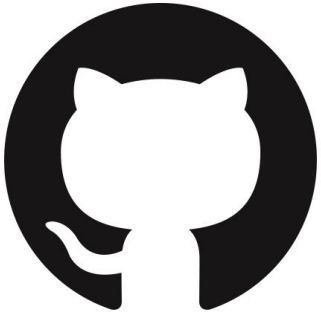
Документ

Ответьте себе или
задайте вопрос

Маршрут вебинара

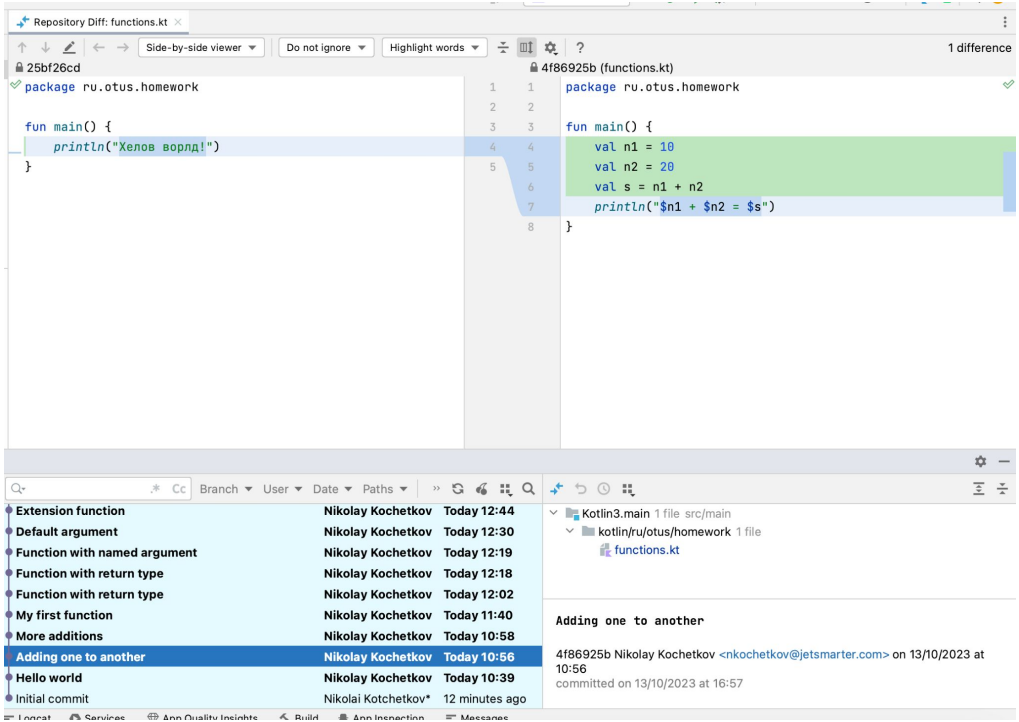
- Выясняем задачи для корутин
- Поход в сеть без корутин
- Корутины - общее
- Callback -> Coroutine
- Обработка ошибок

Репозиторий к занятию

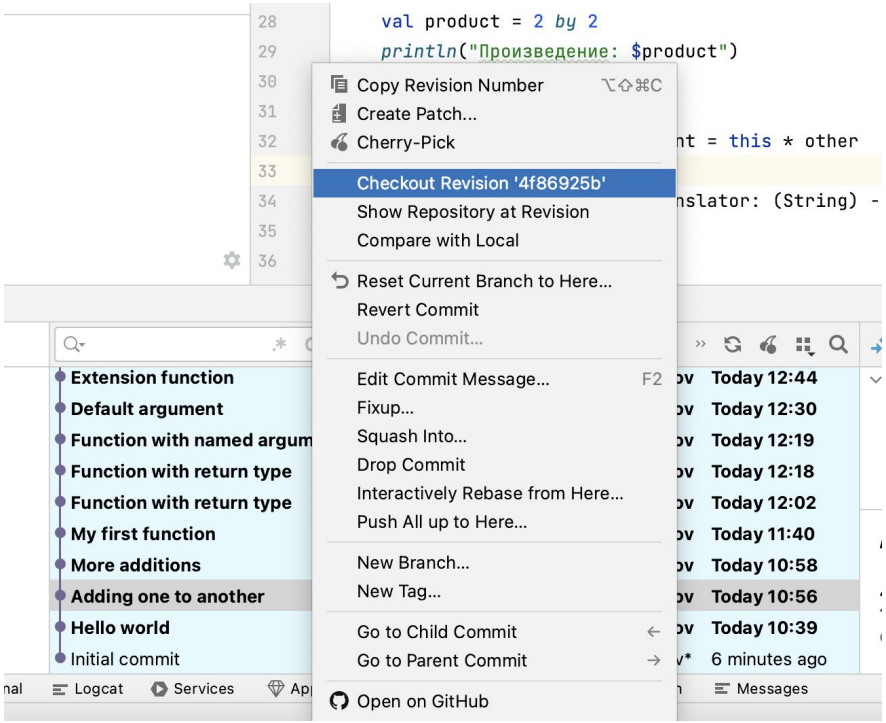


https://github.com/Android-Developer-Basic/Coroutines

Репозиторий к занятию - по шагам



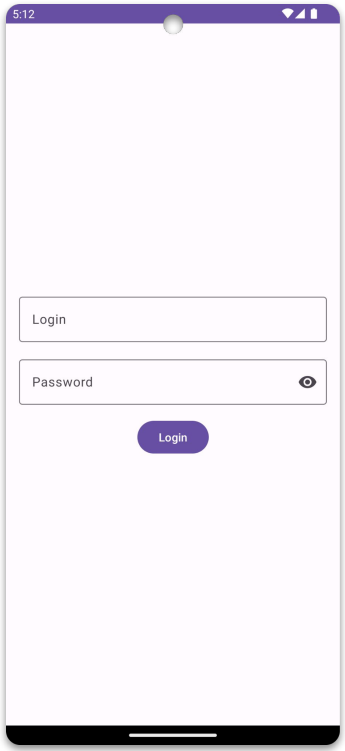
Репозиторий к занятию - по шагам



Приложение с сетью

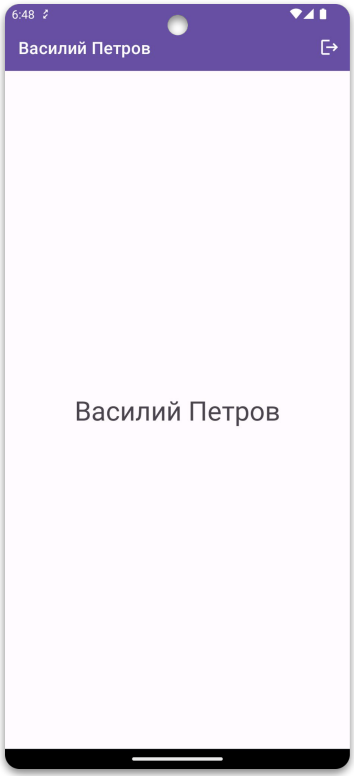
Цель занятия

Приложение



Login

Loading



Content

Приложение

1. Получить от пользователя login/password
2. Обратиться к серверу и получить токен авторизации
3. Получить профиль пользователя

Сетевой сервис

```
/*
 * Базовый URL для нашего сервиса
 */
private const val BASE_URL = "https://my-json-server.typicode.com/Android-Developer-Basic/Coroutines/"

/**
 * API сервера
 */
interface Api {
    @GET("login")
    fun login(@Query("login") login: String, @Query("password") password: String): Call<LoginResponse>

    @GET("profile")
    fun getProfile(@Header("X-Auth-Token") token: String, @Query("id") id: Long): Call<Profile>

    @GET("posts")
    fun getPosts(@Header("X-Auth-Token") token: String): Call<Post>

    companion object {...}
}
```

Сетевой сервис


```
{
  "posts": [
    { "id": 1, "title": "О рыбалке", "created": "2023-11-18T11:43:22.306Z" },
    { "id": 2, "title": "О корутинах", "created": "2023-11-19T11:43:22.306Z" },
    { "id": 3, "title": "О футболе", "created": "2023-11-20T11:43:22.306Z" }
  ],
  "login": {
    "id": 1,
    "token": "12345"
  },
  "profile": {
    "id": 1,
    "name": "Василий Петров",
    "age": 25,
    "registered": "2023-11-17T11:43:22.306Z",
    "interests": [
      "рыбалка", "корутины", "футбол"
    ]
  }
}
```

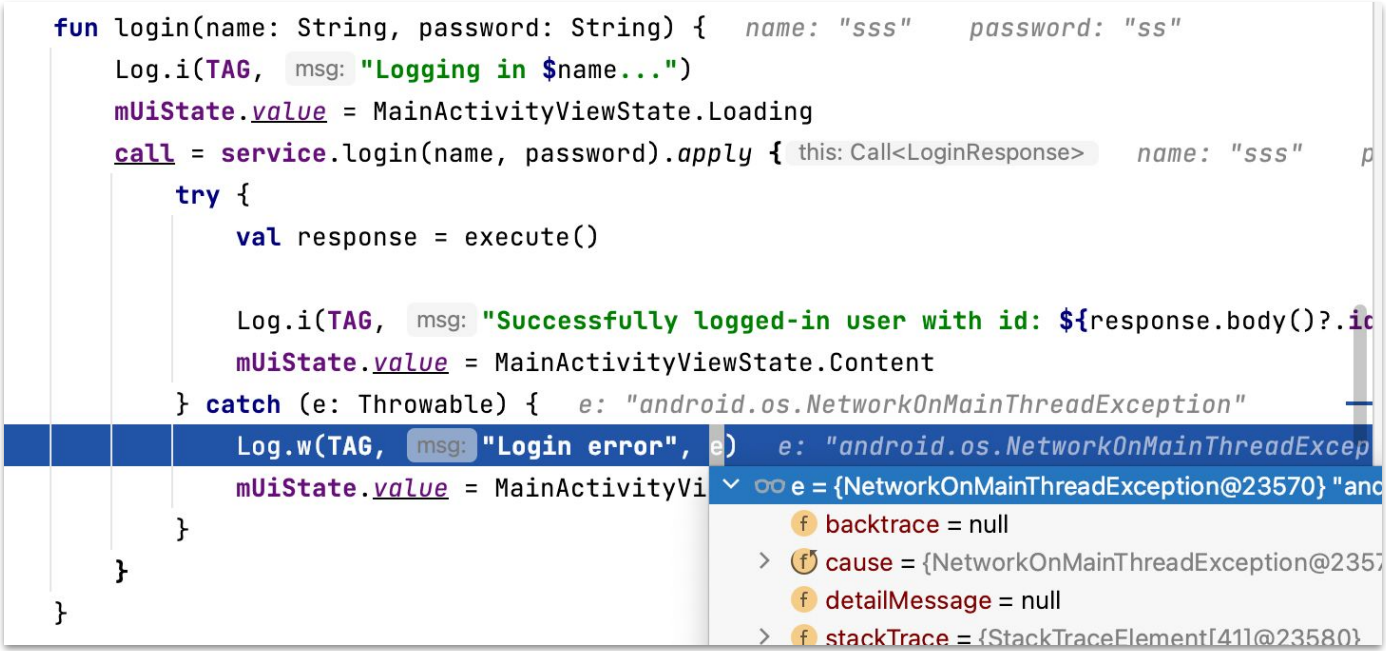
Прямой вызов

Блокирующий вызов в main-потоке

Прямой вызов

```
/**
 * Логин пользователя
 */
fun login(name: String, password: String) {
    Log.i(TAG, msg: "Logging in $name...")
    mUiState.value = MainActivityViewState.Loading
    call = service.login(name, password).apply { this: Call<LoginResponse>
        try {
            val response = execute()

            Log.i(TAG, msg: "Successfully logged-in user with id: ${response.body()?.id}")
            mUiState.value = MainActivityViewState.Content
        } catch (e: Throwable) {
            Log.w(TAG, msg: "Login error", e)
            mUiState.value = MainActivityViewState.Login
        }
    }
}
```

Прямой вызов



В основном потоке нельзя запускать блокирующие операции!

Проблема #1 - блокировка main-потока

Проблема

В основном потоке нельзя запускать блокирующие операции

Решение

Нужен механизм управления потоками

Callbacks

Асинхронные вызовы

Callback - путь счастья

```
/**
 * Логин пользователя
 */
fun login(name: String, password: String) {
    Log.i(TAG, msg: "Logging in $name...")
    mUiState.value = MainActivityViewState.Loading
    call = service.login(name, password).apply { this: Call<LoginResponse>
        enqueue(object : Callback<LoginResponse>{
            override fun onResponse(call: Call<LoginResponse>, response: Response<LoginResponse>) {
                Log.i(TAG, msg: "Successfully logged-in user with id: ${response.body()?.id}")
                mUiState.value = MainActivityViewState.Content
            }

            override fun onFailure(call: Call<LoginResponse>, t: Throwable) {
                Log.w(TAG, msg: "Login error", t)
                mUiState.value = MainActivityViewState.Login
            }
        })
    }
}
```

Будет вызвана, когда мы получим ответ от сервера

Проблема #1 - блокировка main-потока

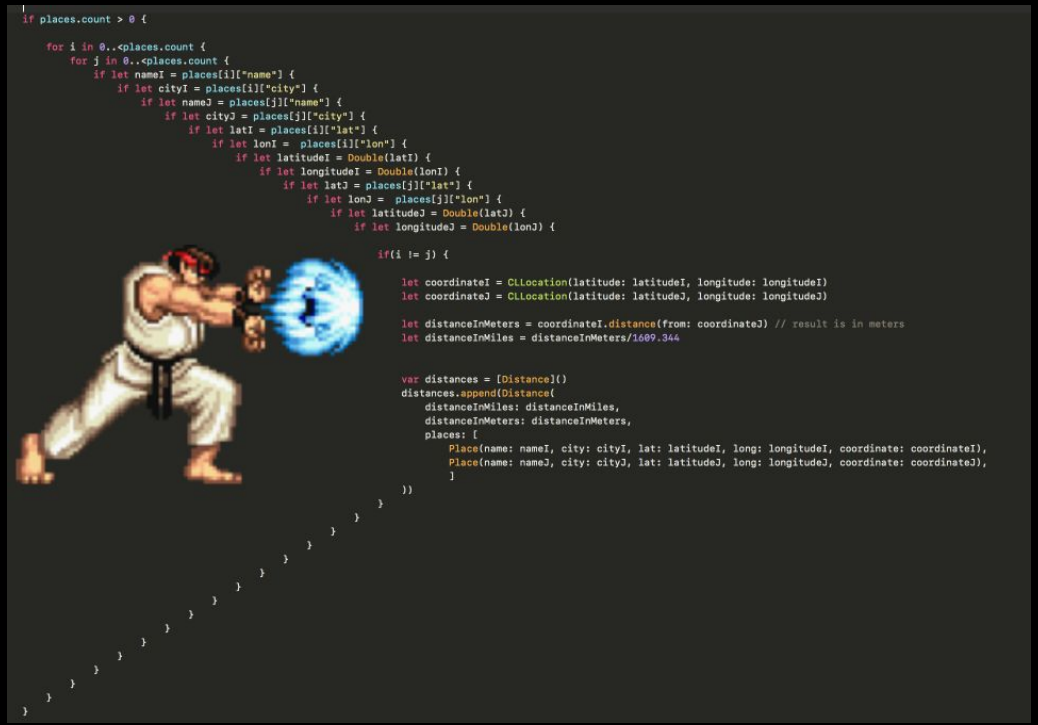
Решение

Решена средствами библиотеки

Добавим загрузку профиля

```
enqueue(object : Callback<LoginResponse>{
    override fun onResponse(call: Call<LoginResponse>, response: Response<LoginResponse>) {
        val loginResponse = response.body()
        if (null == loginResponse) {
            Log.w(TAG, msg: "Empty response!")
            mUiState.value = MainActivityViewState.Login
            return
        }
        Log.i(TAG, msg: "Successfully logged-in user with id: ${loginResponse.id}")

        this@MainActivityViewModel.call = service.getProfile(loginResponse.token, loginResponse.id).apply { this: Call<Profile>
            enqueue(object : Callback<Profile> {
                override fun onResponse(call: Call<Profile>, response: Response<Profile>) {
                    val profile = response.body()
                    if (null == profile) {
                        Log.w(TAG, msg: "No profile!")
                        mUiState.value = MainActivityViewState.Login
                        return
                    }

                    Log.i(TAG, msg: "Successfully loaded profile for: ${profile.name}")
                    mUiState.value = MainActivityViewState.Content(profile.name)
                }

                override fun onFailure(call: Call<Profile>, t: Throwable) {
                    Log.w(TAG, msg: "Profile load error", t)
                    mUiState.value = MainActivityViewState.Login
                }
            })
        }
    }

    override fun onFailure(call: Call<LoginResponse>, t: Throwable) {
        Log.w(TAG, msg: "Login error", t)
        mUiState.value = MainActivityViewState.Login
    }
})
```

Pyramid of Doom



```
if places.count > 0 {

    for i in 0..<places.count {
        for j in 0..<places.count {
            if let nameI = places[i]["name"] {
                if let cityI = places[i]["city"] {
                    if let nameJ = places[j]["name"] {
                        if let cityJ = places[j]["city"] {
                            if let latI = places[i]["lat"] {
                                if let lonI =  places[i]["lon"] {
                                    if let latitudeI = Double(latI) {
                                        if let longitudeI = Double(lonI) {
                                            if let latJ = places[j]["lat"] {
                                                if let lonJ =  places[j]["lon"] {
                                                    if let latitudeJ = Double(latJ) {
                                                        if let longitudeJ = Double(lonJ) {

                                                            if(i != j) {

                                                                let coordinateI = CLLocation(latitude: latitudeI, longitude: longitudeI)
                                                                let coordinateJ = CLLocation(latitude: latitudeJ, longitude: longitudeJ)

                                                                let distanceInMeters = coordinateI.distance(from: coordinateJ) // result is in meters
                                                                let distanceInMiles = distanceInMeters/1609.344


                                                                var distances = [Distance]()
                                                                distances.append(Distance(
                                                                    distanceInMiles: distanceInMiles,
                                                                    distanceInMeters: distanceInMeters,
                                                                    places: [
                                                                        Place(name: nameI, city: cityI, lat: latitudeI, long: longitudeI, coordinate: coordinateI),
                                                                        Place(name: nameJ, city: cityJ, lat: latitudeJ, long: longitudeJ, coordinate: coordinateJ),
                                                                        ]
                                                                ))
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Проблема #2 - читаемость кода

Проблема

Callback-подход увеличивает вложенность и читаемость

Решение

Нужен способ “выровнять” исходный код

Проблема #3 - обработка ошибок

Проблема

Как правильно обработать ошибку при сильном уровне вложенности?

Решение

Создать схему, близкую к try-catch

Проблема #4 - время жизни операции

Проблема

Результат работы асинхронной операции может прийти после нашей смерти

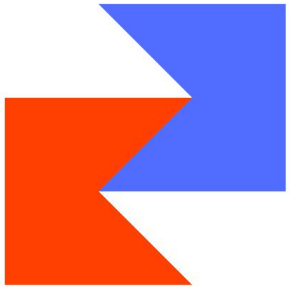
Решение

Предусмотреть отмену операции

Проблемы

1. Управление потоками
2. Читаемость кода
3. Обработка ошибок
4. Время жизни операции

Вопросы?



Kotlin Coroutines

Что такое корутина?

```
fun main() = runBlocking { this: CoroutineScope // this: CoroutineScope
    launch { this: CoroutineScope // Запустить корутину и продолжить
        delay( timeMillis: 1000L) // Не-блокирующая задержка
        println("World!") // Напечатать после задержки
    }
    println("Hello") // Основная корутина продолжает работать
}
```

```
Hello
World!
```

Что такое корутина?

“Корутина - запущенный экземпляр асинхронного вычисления”

Structured concurrency

- Корутина запускается внутри *CoroutineScope*
- Scope определяет жизненный цикл корутины
- Внешний scope завершается только когда завершатся дочерние
- Ошибка в дочернем scope вызовет ошибку в родительском (по умолчанию)

Вопросы?

Suspend function

```
fun main() = runBlocking { this: CoroutineScope // this: CoroutineScope
    launch { doWorld() }
    println("Hello")
}

// Функция, которая может встать на паузу
suspend fun doWorld() {
    delay( timeMillis: 1000L)
    println("World!")
}
```

Suspend функция

```
44  /**
45   * Процедура логина
46   */
47  class LoginUseCase {
48      suspend fun loginUser(userId: String, password: String): User {
49          val user = userRemoteDataSource.logUserIn(userId, password)
50          val userDb = userLocalDataSource.logUserIn(user)
51          return userDb
52      }
53  }
```

Suspend функция (Kotlin bytecode)

“Продолжение”

```
@Nullable
public final Object logUserIn(@NotNull String var1, @NotNull String var2, @NotNull Continuation var3) {
```

Continuation

```
Interface representing a continuation after a suspension point that returns a value of type T.
@SinceKotlin( version: "1.3")
public interface Continuation<in T> {
    The context of the coroutine that corresponds to this continuation.
    public val context: CoroutineContext

    Resumes the execution of the corresponding coroutine passing a successful or failed result as the return
    value of the last suspension point.
    public fun resumeWith(result: Result<T>)
}
```

Continuation

```
84  /**
85   * Условно, это выглядит так
86   */
87  fun loginUser(userId: String, password: String, completion: Continuation<Any?>) {
88      val user = userRemoteDataSource.logUserIn(userId, password)
89      val userDb = userLocalDataSource.logUserIn(user)
90      completion.resume(userDb)
91  }
```

Suspension points

```
83  /**
84   * Условно, это выглядит так
85   */
86  fun loginUser(userId: String, password: String, completion: Continuation<Any?>) {
87      // Label 0 -> первый вызов
88      val user = userRemoteDataSource.logUserIn(userId, password)
89      // Label 1 -> возврат после сетевого запроса
90      val userDb = userLocalDataSource.logUserIn(user)
91      // Label 2 -> возврат после локального запроса
92      completion.resume(userDb)
93  }
```

Suspension points

```
/**
 * Условно, это выглядит так
 */
fun loginUser(userId: String, password: String, completion: Continuation<Any?>) {
    when(label) {
        0 -> { // Label 0 -> первый вызов
            userRemoteDataSource.logUserIn(userId, password)
        }
        1 -> { // Label 1 -> возврат после сетевого запроса
            userLocalDataSource.logUserIn(user)
        }
        2 -> { // Label 2 -> возврат после локального запроса
            completion.resume(userDb)
        }
        else -> throw IllegalStateException()
    }
}

```

Continuation - коммуникация между suspend функциями!

Callback -> Suspend fun

```
116  /**
117   * Переход из колбеков в корутины
118   */
119  suspend fun callDaData(service: DaDataService): Suggestion = suspendCancellableCoroutine { cont ->
120      // Создаем запрос в сеть
121      val call = service.getAddressHint(QueryString( query: "aaaaaaa"))
122      // Этот колбек вызовется, когда запрос вернет результат
123      val callback = object : Callback<Suggestion> {
124          override fun onResponse(call: Call<Suggestion>, response: Response<Suggestion>) {
125              // Все хорошо
126              cont.resume(response.body()!!)
127          }
128          override fun onFailure(call: Call<Suggestion>, t: Throwable) {
129              // Все плохо
130              cont.resumeWithException(t)
131          }
132      }
133      // Запускаем запрос
134      call.enqueue(callback)
135
136      // Если пользователю больше не нужно...
137      cont.invokeOnCancellation { it: Throwable?
138          call.cancel()
139      }
140  }
```

Вопросы?

Job

“Корутина - запущенный экземпляр асинхронного вычисления”

Job

```
25  fun main() = runBlocking {  this: CoroutineScope
26      val job = launch {  this: CoroutineScope
27          delay( timeMillis: 1000L)
28          println("World!")
29      }
30      println("Hello")
31      job.join() // Ждать, пока завершится дочерняя задача
32      println("Done")
33  }
```

```
Hello
World!
Done
```

Job - отмена

```
25  fun main() = runBlocking { this: CoroutineScope
26      val job = launch { this: CoroutineScope
27          repeat( times: 1000) { i ->
28              println("job: I'm sleeping $i ...")
29              delay( timeMillis: 500L)
30          }
31      }
32      delay( timeMillis: 1300L)
33      println("main: I'm tired of waiting!")
34      job.cancel() // Отменить задачу
35      job.join() // Ждать окончания
36      println("main: Now I can quit.")
37  }
```

```
job: I'm sleeping 0 ...
job: I'm sleeping 1 ...
job: I'm sleeping 2 ...
main: I'm tired of waiting!
main: Now I can quit.
```

Вопросы?

Корутины - порядок выполнения

```
suspend fun doSomethingUsefulOne(): Int {
    delay( timeMillis: 1000L)
    return 13
}

suspend fun doSomethingUsefulTwo(): Int {
    delay( timeMillis: 1000L)
    return 29
}
```

Корутины - последовательное выполнение

```
fun main() = runBlocking { this: CoroutineScope
    val time = measureTimeMillis {
        val one = doSomethingUsefulOne()
        val two = doSomethingUsefulTwo()
        println("The answer is ${one + two}")
    }
    println("Completed in $time ms")
}
```

```
The answer is 42
Completed in 2017 ms
```

Async/await - параллельное выполнение

```
fun main() = runBlocking { this: CoroutineScope
    val time = measureTimeMillis {
        val one = async { doSomethingUsefulOne() }
        val two = async { doSomethingUsefulTwo() }
        println("The answer is ${one.await() + two.await()}")
    }
    println("Completed in $time ms")
}
```

```
The answer is 42
Completed in 1017 ms
```

Вопросы?

CoroutineContext - контекст выполнения

- Корутины всегда исполняются в контексте (и в scope)
- Контекст состоит из элементов: Job, Dispatcher, Имя
- Контекст внутренней корутины наследует контекст внешней (по умолчанию)

Dispatcher - определяет поток выполнения

```
@OptIn(ExperimentalCoroutinesApi::class)
fun main(): Unit = runBlocking { this: CoroutineScope
    launch { this: CoroutineScope // контекст родителя
        println("main runBlocking      : I'm working in thread ${Thread.currentThread().name}")
    }
    launch(Dispatchers.Unconfined) { this: CoroutineScope // без диспетчеризации
        println("Unconfined            : I'm working in thread ${Thread.currentThread().name}")
    }
    launch(Dispatchers.Default) { this: CoroutineScope // ограниченный пул потоков
        println("Default               : I'm working in thread ${Thread.currentThread().name}")
    }
    launch(newSingleThreadContext( name: "MyOwnThread")) { this: CoroutineScope // запускает новый Thread
        println("newSingleThreadContext: I'm working in thread ${Thread.currentThread().name}")
    }
}
```

Dispatcher - определяет поток выполнения

```
Unconfined      : I'm working in thread main
main runBlocking: I'm working in thread main
Unconfined      : After delay in thread kotlinx.coroutines.DefaultExecutor
main runBlocking: After delay in thread main
```

Вопросы?

Исключения - явный try/catch

```
suspend fun badFun() {
    delay( timeMillis: 100)
    throw RuntimeException("Error")
}

fun main(): Unit = runBlocking { this: CoroutineScope
    launch { this: CoroutineScope
        try {
            withContext(Dispatchers.IO)  { this: CoroutineScope
                badFun()
            }
        } catch (e: Throwable) {
            println("Caught: $e")
        }
    }
}
```

```
Caught: java.lang.RuntimeException: Error
```

Исключения - exception handler

```
val handler : CoroutineExceptionHandler = CoroutineExceptionHandler { _, exception ->
    println("CoroutineExceptionHandler: $exception")
}

fun main(): Unit = runBlocking { this: CoroutineScope
    val cs = CoroutineScope(newSingleThreadContext( name: "t1"))
    val j1 = cs.launch(handler) { this: CoroutineScope
        delay( timeMillis: 1000)
        throw RuntimeException("error!")
    }
    j1.join()
}
```

```
CoroutineExceptionHandler: java.lang.RuntimeException: error!
```

Вопросы?

Demo

Рефлексия

Маршрут вебинара

- Выясняем задачи для корутин
- Поход в сеть без корутин
- Корутины - общее
- Callback -> Coroutine
- Обработка ошибок

Вопросы для проверки

По пройденному материалу всего вебинара

1. Какие проблемы решают корутины?
2. Что такое CoroutineScope?
3. Что такое CoroutineContext и чем он отличается от CoroutineScope?

Рефлексия

С какими впечатлениями уходите с вебинара?

Как будете применять на практике то,
что узнали на вебинаре?

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Спасибо за внимание!

Приходите на следующие вебинары

Николай Кочетков

Руководитель Андроид разработки FlyXO

Об опыте (например):

22 года в IT, 7 лет - играющий тренер Андроид

Телефон / эл. почта / соц. сети:
LinkedIn: https://www.linkedin.com/in/motorro/
GitHub: https://github.com/motorro/
Medium: https://medium.com/@motorro