**Пример Coroutines**

В учебном пособии представлены ключевые понятия выполнения асинхронных задач в приложениях Android с использованием корутин. Теперь на основе этих знаний, создадим пример приложения, которое запускает тысячи корутинов одним нажатием кнопки.

**1.Создание примера приложения Coroutine**

Выберите параметр *New Project* на экране приветствия и в появившемся диалоговом окне нового проекта выберите шаблон «Empty Activity», прежде чем нажать кнопку «Далее».

Введите *CoroutineDemo* в поле Name. Прежде чем нажать кнопку Готово, измените настройку Минимальный уровень API на API 26: Android 8.0 (Oreo) и выберите язык Kotlin. Добавьте в проект возможности ViewBinding.

**2.Добавление поддержки Coroutine в проект**

Текущая версия Android Studio не включает автоматически поддержку корутинов во вновь создаваемых проектах. Поэтому, прежде чем продолжить, отредактируйте и добавьте следующие строки в раздел зависимостей (отмечая, как всегда, что могут быть доступны более новые версии библиотек):

**implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.5.2'**

**implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.5.2'**

После внесения изменений нажмите на ссылку *SyncNow* в верхней части панели редактора, чтобы зафиксировать изменения.

**3.Проектирование пользовательского интерфейса**

Пользовательский интерфейс будет состоять из кнопки запуска корутинов вместе с Seekbar, чтобы указать, сколько корутинов должно быть запущено асинхронно при каждом нажатии кнопки. По мере выполнения корутинов TextView будет обновляться при запуске и окончании отдельных корутин.

Начните с загрузки файла макета и добавьте объекты Button, TextView и SeekBar, чтобы макет напоминал тот, который показан на рисунке 1:

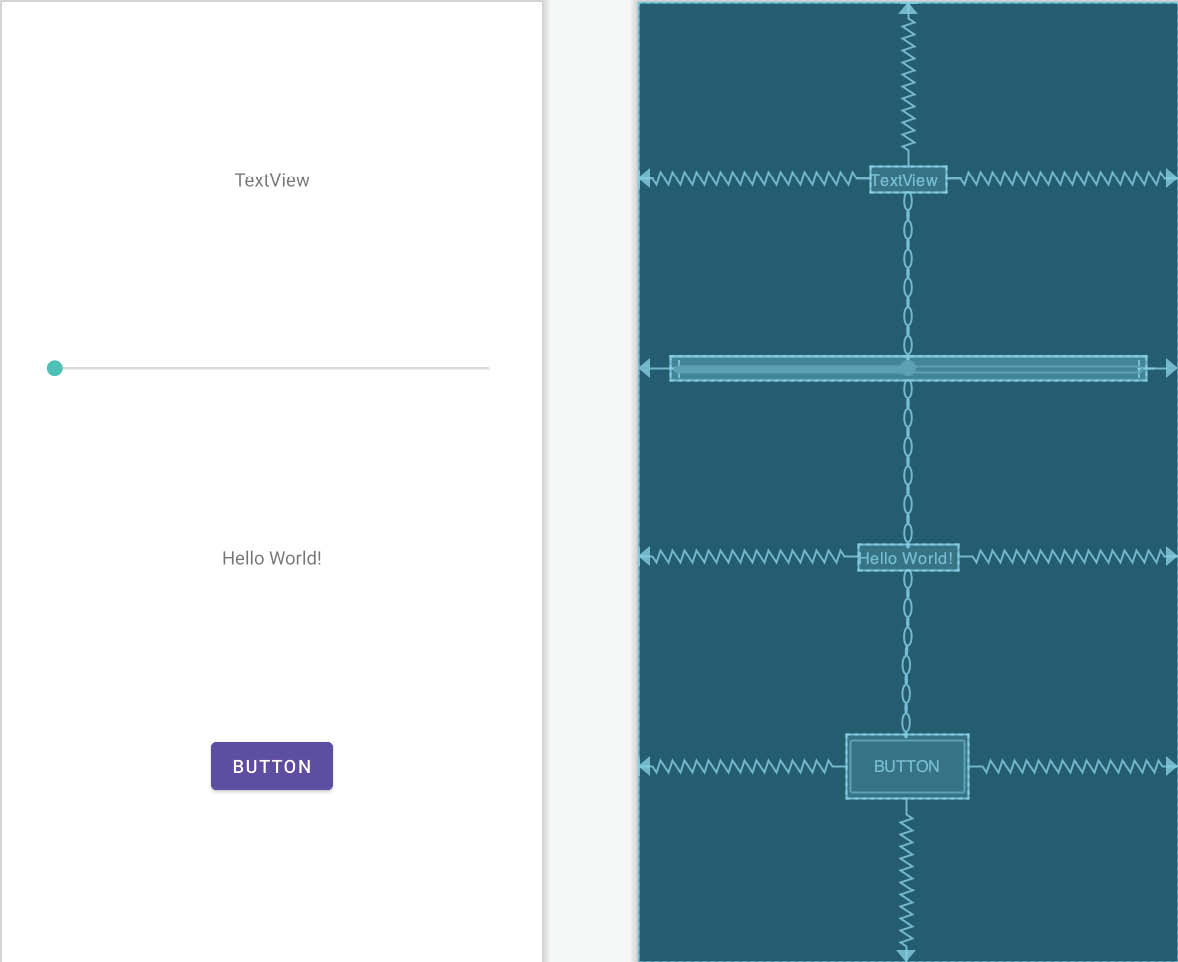
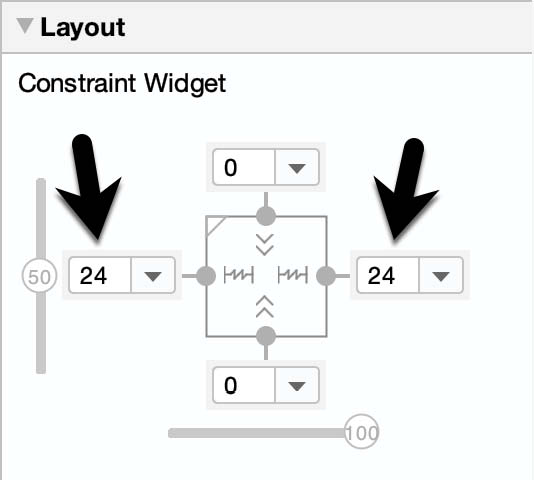


Рисунок 1

Чтобы реализовать ограничения макета, показанные выше, начните с очистки всех ограничений на макете с помощью кнопки панели инструментов. Щелкните на четыре объекта, удерживая нажатой клавишу «Shift», чтобы все они были выбраны, щелкните правой кнопкой мыши самый верхний элемент управления TextView и выберите пункт меню *Center -> Horizontally*. Щелкните правой кнопкой мыши еще раз, на этот раз выбрав опцию *Chains -> Create Vertical Chain*.

Выберите SeekBar и измените свойство layout\_width на 0dp (match\_constraints), прежде чем добавлять поле 24dp с левой и правой стороны, как показано на рисунке 2.

Измените атрибут onClick для Button, чтобы вызвать метод с именем *launchCoroutines* и измените идентификаторы самого верхнего элемента TextView, SeekBar и нижнего элемента TextView на *countText*, *seekBar* и *statusText* соответственно. Наконец, измените текст на кнопке на «Запуск корутин» и извлеките текст в строковый ресурс.

**4.Реализация SeekBar**

SeekBar управляет количеством асинхронных корутин, от 1 до 2000, которые запускаются при каждом нажатии кнопки. Оставшись в файле *activity\_main.xml*, выберите SeekBar и с помощью окна инструментов Атрибуты измените свойство max на 2000. Затем отредактируйте файл *MainActivity.kt* добавьте переменную, в которой будет храниться текущий параметр ползунка, и измените метод *onCreate(),* чтобы добавить слушатель SeekBar:

class MainActivity : AppCompatActivity() {

private lateinit var binding: ActivityMainBinding

**private var count: Int = 1**

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

binding = ActivityMainBinding.inflate(layoutInflater)

setContentView(binding.root)

**binding.seekBar.setOnSeekBarChangeListener(object :**

**SeekBar.OnSeekBarChangeListener {**

**override fun onProgressChanged(seek: SeekBar,**

**progress: Int, fromUser: Boolean) {**

**count = progress**

**binding.countText.text = "${count} coroutines"**

**}**

**override fun onStartTrackingTouch(seek: SeekBar) {**

**}**

**override fun onStopTrackingTouch(seek: SeekBar) {**

**}**

**})**

}

Когда указатель ползунка сдвигается, текущее значение будет сохранено в переменной count и отображено в представлении countText.

**5.Добавление suspend функции**

Когда пользователь нажмет кнопку, приложению потребуется запустить количество корутин, выбранных в SeekBar. Метод onClick позволит достичь этого с помощью конструктора корутин для выполнения suspend функции. Поскольку функция приостановки возвращает строку состояния, отображаемую в объекте statusText TextView, ее необходимо будет реализовать с помощью строителя *async*. Все эти действия должны быть выполнены в рамках корутина, который также должен быть объявлен. В файл внесите следующие изменения:

class MainActivity : AppCompatActivity() {

**private val coroutineScope = CoroutineScope(Dispatchers.Main)**

**suspend fun performTask(tasknumber: Int): Deferred<String> =**

**coroutineScope.async(Dispatchers.Main) {**

**delay(5\_000)**

**return@async "Finished Coroutine ${tasknumber}"**

**}**

Учитывая, что функция выполняет лишь небольшую задачу и предполагает изменения пользовательского интерфейса, корутин выполняется с помощью главного диспетчера. Передается порядковый номер запускаемого корутина, он задерживается на 5 секунд, а затем возвращает строку, указывающую, что пронумерованный корутин завершён.

**6.Реализация метода запуска Coroutines**

Последней задачей перед тестированием приложения является добавление метода который вызывается при нажатии на объект Button. Этот метод должен быть добавлен в файл *MainActivity.kt* следующим образом:

**fun launchCoroutines(view: View) {**

**(1..count).forEach {**

**binding.statusText.text = "Started Coroutine ${it}"**

**coroutineScope.launch(Dispatchers.Main) {**

**binding.statusText.text = performTask(it).await()**

**}**

**}**

**}**

Метод реализует цикл для запуска запрошенного количества корутинов и обновляет состояние TextView каждый раз, когда результат возвращается из завершенного корутина с помощью вызова метода *await()*.

**7.Тестирование приложения**

Создайте и запустите приложение на устройстве или эмуляторе и переместите SeekBar на низкое число (например, 10), прежде чем нажать кнопку запуска. Текст состояния будет обновляться каждый раз при запуске корутина до тех пор, пока не будет достигнут максимум. После того, как каждый корутин завершит 5-секундную задержку, текст статуса будет обновляться до тех пор, пока все 10 не будут завершены (на практике эти обновления статуса будут происходить так быстро, что будет трудно увидеть изменения статуса).

Повторите процесс с seekBar, установленным в значение 2000, на этот раз перемещая указатель вперед и назад по мере запуска корутинов, чтобы убедиться, что основной поток все еще работает и не заблокирован.

Наконец, с отображением панели Logcat установите для SeekBar значение 2000 и нажмите кнопку запуска несколько раз. Примерно через 15 щелчков на панели Logcat начнут отображаться сообщения, подобные приведенным ниже:

I/Choreographer: Skipped 52 frames! The application may be doing too much work on its main thread.

Пропустил 52 кадра! Возможно, приложение выполняет слишком много работы над своим основным потоком.

Хотя приложение продолжает функционировать, очевидно, что объем корутинов, работающих внутри приложения, начинает перегружать основной поток. Тот факт, что это происходит только тогда, когда десятки тысяч корутинов выполняются одновременно, является свидетельством эффективности корутинов Kotlin. Однако, когда это сообщение начинает появляться в ваших собственных приложениях, это может быть признаком того, что выполняется слишком много корутин, либо выполняемая асинхронная рабочая нагрузка слишком тяжела для основного потока. В этом случае может потребоваться использовать другой диспетчер, возможно, используя конструктор *withContext*.

**Резюме**

Основываясь на изученной информации, мы рассмотрели создание примера приложения, демонстрирующего использование корутинов Kotlin в приложении для Android. Пример продемонстрировал использование главного диспетчера для запуска тысяч асинхронных корутин, включая возвращаемые результаты.