Лекция #30. Потоки, фрагменты и ViewModel

При использовании вторичных потоков следует учитывать следующий момент. Более оптимальным способом является работа потоков с фрагментом, нежели непосредственно с activity. Например, определим в файле activity_main.xml следующий интерфейс:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textViewStatus"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="4dp"
        android:text="Статус"
        app:layout_constraintBottom_toTopOf="@+id/progressBar"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent" />

    <Button
        android:id="@+id/buttonLaunch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp"
        android:layout_marginBottom="16dp"
        android:text="Запуск"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <ProgressBar
        android:id="@+id/progressBar"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
```

```
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp"
        app:layout_constraintBottom_toTopOf="@+id/buttonLaunch"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

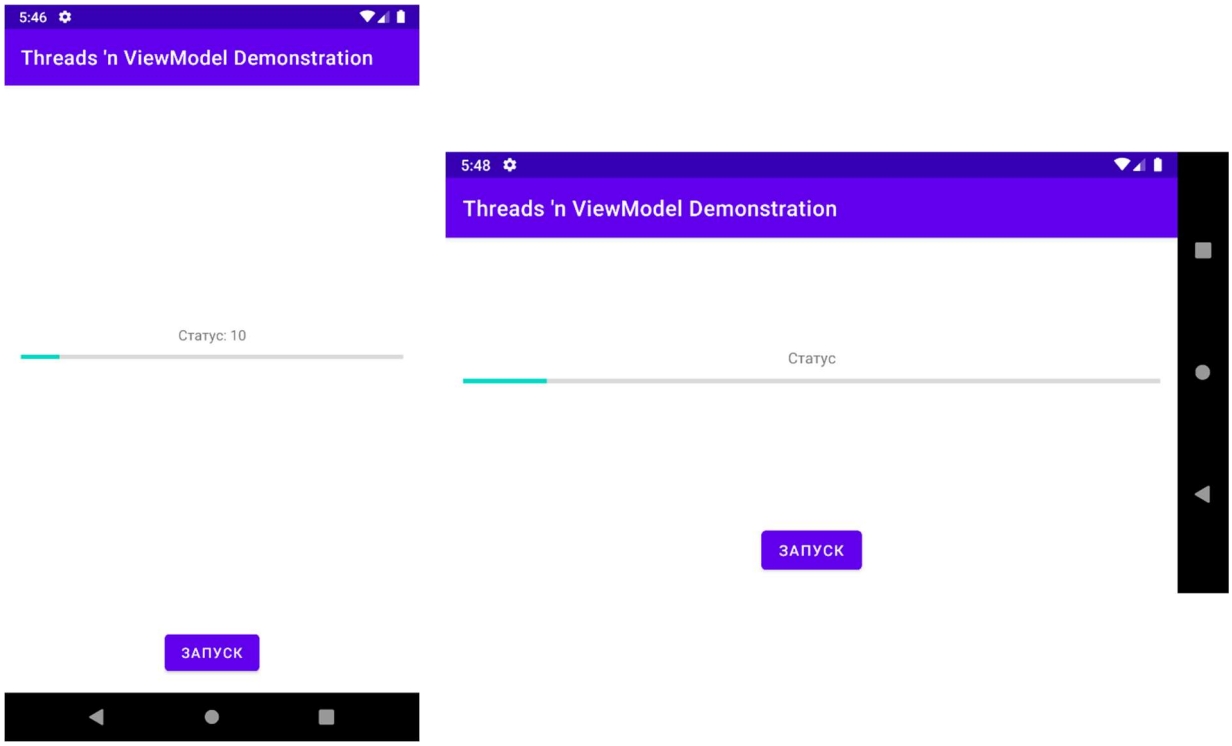
Здесь определена кнопка для запуска вторичной задачи и элементы **TextView** и **ProgressBar**, которые отображают индикацию выполнения задачи.

В классе **MainActivity** определим следующий код:

```
package com.awkitsune.threadsnviewmodeldemonstration

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.ProgressBar
import android.widget.TextView
import java.lang.Exception

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val progressBar: ProgressBar = findViewById(R.id.progressBar)
        val textViewStatus: TextView = findViewById(R.id.textViewStatus)
        val button: Button = findViewById(R.id.buttonLaunch)
        button.setOnClickListener(){
            var runnable = Runnable {
                for (currentValue in 0..100){
                    try {
                        textViewStatus.post(Runnable {
                            progressBar.progress = currentValue
                            textViewStatus.text = "Статус: $currentValue"
                        })
                        Thread.sleep(500)
                    }
                    catch (e: InterruptedException){
                        e.printStackTrace()
                    }

                }
            }

            val thread = Thread(runnable)
            thread.start()
        }
    }
}
```

Здесь по нажатию кнопки мы запускаем задачу Runnable, в которой в цикле от 0 до 100 изменяем показатели ProgressBar и TextView, имитируя некоторую долгую работу.

Однако если в процессе работы задачи мы изменим ориентацию мобильного устройства, то произойдет пересоздание activity, и приложение перестанет работать должным образом.



В данном случае проблема упирается в состояние, которым оперирует поток, а именно - переменную **currentValue**, к значению которой привязаны виджеты в Activity.

Добавление ViewModel

Для подобных случаев в качестве решения проблемы предлагается использовать ViewModel. Итак, добавим в ту же папку, где находится файл **MainActivity.kt**, новый класс **MainViewModel** со следующим кодом:

```
package com.awkitsune.threadsnviewmodeldemonstration

import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;
import androidx.lifecycle.ViewModel;

class MainViewModel: ViewModel() {
    private val isStarted = MutableLiveData(false)
    private var value: MutableLiveData<Int>? = null

    fun getValue(): LiveData<Int>? {
        if (value == null) {
            value = MutableLiveData(0)
        }
        return value
    }
```

```
    fun execute() {
        if (!isStarted.value!!) {
            isStarted.postValue(true)
            val runnable = Runnable {
                for (i in value!!.value!!..100) {
                    try {
                        value!!.postValue(i)
                        Thread.sleep(400)
                    }
                    catch (e: InterruptedException) {
                        e.printStackTrace()
                    }
                }
            }
            val thread = Thread(runnable)
            thread.start()
        }
    }
}
```

Итак, здесь определен класс **MyViewModel**, который унаследован от класса **ViewModel**, специально предназначенного для хранения и управления состоянием или моделью.

В качестве состояния здесь определены для объекта. В первую очередь, это числовое значение, к которым будут привязан виджеты MainActivity. И во-вторых, нам нужен некоторый индикатор того, что поток уже запущен, чтобы по нажатию на кнопку не было запущено лишних потоков.

Для хранения числового значения предназначена переменная value:

```
private var value: MutableLiveData<Int>? = null
```

Для привязки к этому значению оно имеет тип **MutableLiveData**. А поскольку мы будем хранить в этой переменной числовое значение, то тип переменной типизирован типом **Integer**.

Для доступа извне класса к этому значению определен метод **getValue**, который имеет тип **LiveData** и который при первом обращении к переменной устанавливает 0, либо просто возвращает значение переменной:

```
fun getValue(): LiveData<Int>? {
    if (value == null) {
        value = MutableLiveData(0)
    }
    return value
}
```

Для индикации, запущен ли поток, определена переменная **isStarted**, которая хранит значение типа **Boolean**, то есть фактически **true** или **false**. По умолчанию она имеет значение false (то есть поток не запущен).

Для изменения числового значения, к которому будут привязаны виджеты, определен метод **execute()**. Он запускает поток, если поток не запущен:

```
if (!isStarted.value!!)
```

Далее переключает значение переменной **isStarted** на **true**, поскольку мы запускаем поток.

В потоке также запускается цикл:

```
for (i in value!!.value!!..100)
```

И в данном случае мы пользуемся преимуществом класса **ViewModel**, который позволяет автоматически сохранять свое состояние.

Причем счетчик цикла в качестве начального значения берет значение из переменной value и увеличается на единицу, пока не дойдет до ста.

В самом цикле изменяется значение переменной **value** с помощью передачи значения в метод **postValue()**:

```
value!!.postValue(i)
```

Таким образом, в цикле осуществится проход от 0 до 100, и при каждой итерации цикла будет изменяться значение переменной value.

Теперь задействуем наш класс MyViewModel и для этого изменим код класса **MainActivity**:

```
package com.awkitsune.threadsnviewmodeldemonstration

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.ProgressBar
import android.widget.TextView
import androidx.lifecycle.ViewModelProvider
import java.lang.Exception

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val progressBar: ProgressBar = findViewById(R.id.progressBar)
        val textViewStatus: TextView = findViewById(R.id.textViewStatus)
        val button: Button = findViewById(R.id.buttonLaunch)

        val mainViewModel: MainViewModel =
ViewModelProvider(this).get(MainViewModel::class.java)
```

```
        mainViewModel.getValue()?.observe(this) { value ->
            progressBar.progress = value
            textViewStatus.setText("Статус: $value")
        }

        button.setOnClickListener { mainViewModel.execute() }
    }
}
```

Чтобы задействовать MyViewModel, создаем объект класса **ViewModelProvider**, в конструктор которого передается объект-владелец ViewModel. В данном случае это текущий объект MainActivity:

```
ViewModelProvider(this)
```

И далее с помощью метода **get()** создаем объект класса ViewModel, который будет использоваться в объекте MainActivity:

```
val mainViewModel: MainViewModel =
        ViewModelProvider(this).get(MainViewModel::class.java)
```

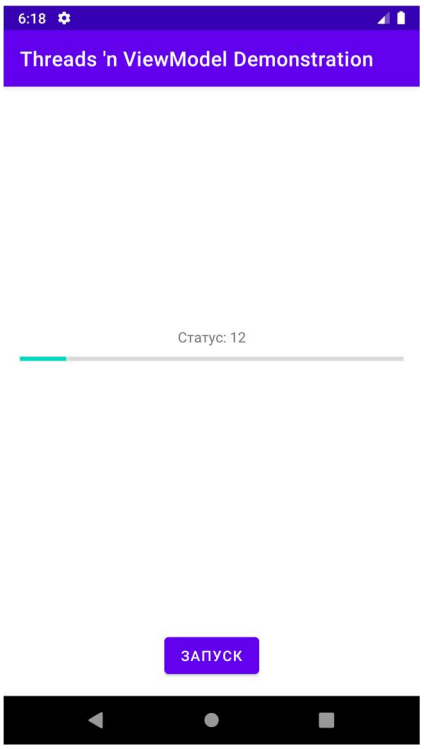
Получив объект MyViewModel, определяем прослушивание изменений его переменной value с помощью метода **observe**:

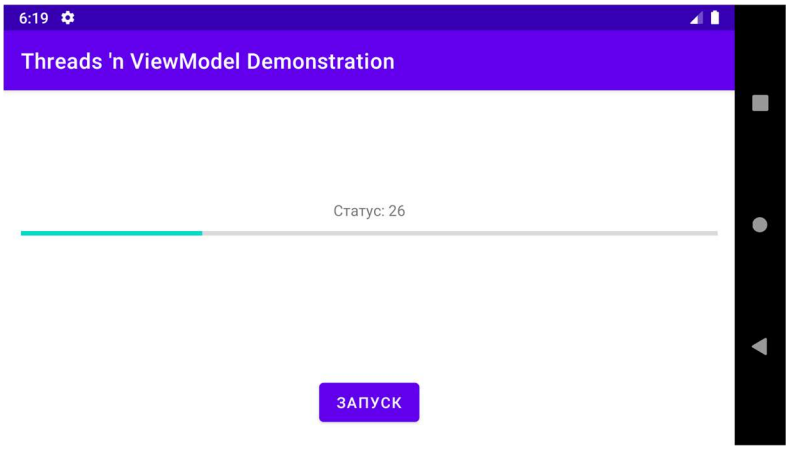
```
mainViewModel.getValue()?.observe(this) { value ->
    progressBar.progress = value
    textViewStatus.setText("Статус: $value")
}
```

Метод **observe()** в качестве первого параметра принимает владельца функции обсервера - в данном случае текущий объект **MainActivity**. А в качестве второго параметра - функцию обсервера (а точнее объект интерфейса Observer). Функция обсервера принимает один параметр - новое значение отслеживаемой переменной (то есть в данном случае переменной value). Получив новое значение переменной value, мы изменяем параметры виджетов.

Таким образом, при каждом изменении значения в переменной value виджеты получат ее новое значение.

Теперь если мы запустим приложение, то вне зависимости от смены ориентации моильного устройства фоновая задача будет продолжать свою работу:


6:18
Threads 'n ViewModel Demonstration
Статус: 12
ЗАПУСК


6:19
Threads 'n ViewModel Demonstration
Статус: 26
ЗАПУСК