Лекция #29. Многопоточность и асинхронность. Создание потоков и визуальный интерфейс

Когда мы запускаем приложение на Android, система создает поток, который называется основным потоком приложения или UI-поток. Этот поток обрабатывает все изменения и события пользовательского интерфейса. Однако для вспомогательных операций, таких как отправка или загрузка файла, продолжительные вычисления и т.д., мы можем создавать дополнительные потоки.

Для создания новых потоков нам доступен стандартный функционал класса **Thread** из базовой библиотеки **Java** из пакета **java.util.concurrent**, которые особой трудности не представляют. Тем не менее трудности могут возникнуть при обновлении визуального интерфейса из потока.

Например, создадим простейшее приложение с использованием потоков. Определим следующую разметку интерфейса в файле **activity_main.xml**:

```
<?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/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:layout_marginBottom="16dp"
        android:gravity="center"
```

```
        android:text="Hello World!"
        app:layout_constraintBottom_toTopOf="@+id/button"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp"
        android:layout_marginBottom="16dp"
        android:text="Запустить поток"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

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

Здесь определена кнопка для запуска фонового потока, а также текстовое поле для отображения некоторых данных, которые будут генерироваться в запущенном потоке.

Далее определим в классе **MainActivity** следующий код:

```
package com.awkitsune.multithreadingdemonstration

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.TextView
import java.util.*

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

        val textView: TextView = findViewById(R.id.textView)
        val button: Button = findViewById(R.id.button)
        button.setOnClickListener(){
            var runnable: Runnable = Runnable {
                val calendar: Calendar = Calendar.getInstance()
                val hours = calendar.get(Calendar.HOUR_OF_DAY)
                val minutes = calendar.get(Calendar.MINUTE)
                val seconds = calendar.get(Calendar.SECOND)

                textView.text = "$hours : $minutes : $seconds"
            }
            val thread: Thread = Thread(runnable)
            thread.start()
        }
    }
}
```

Итак, здесь к кнопке прикреплен обработчик нажатия, который запускает новый поток. Создавать и запускать поток в Java можно различными

способами. В данном случае сами действия, которые выполняются в потоке, неявно определяются в методе **run()** объекта **Runnable**:

```
var runnable: Runnable = Runnable {
    val calendar: Calendar = Calendar.getInstance()
    val hours = calendar.get(Calendar.HOUR_OF_DAY)
    val minutes = calendar.get(Calendar.MINUTE)
    val seconds = calendar.get(Calendar.SECOND)

    textView.text = "$hours : $minutes : $seconds"
}
```

Для примера получаем текущее время и пытаемся отобразить его в элементе TextView.

Далее определяем объект потока - объект **Thread**, который принимает объект Runnable. И с помощью метода **start()** запускаем поток:

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

Вроде ничего сложного. Но если мы запустим приложение и нажмем на кнопку, то мы столкнемся с ошибкой:

```
E/AndroidRuntime: FATAL EXCEPTION: Thread-2
    Process: com.awkitsune.multithreadingdemonstration, PID: 18472
    android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.
        at android.view.ViewRootImpl.checkThread(ViewRootImpl.java:7753)
        at android.view.ViewRootImpl.requestLayout(ViewRootImpl.java:1225)
        at android.view.View.requestLayout(View.java:23093)
        at android.view.View.requestLayout(View.java:23093)
        at android.view.View.requestLayout(View.java:23093)
        at android.view.View.requestLayout(View.java:23093)
        at android.view.View.requestLayout(View.java:23093)
        at android.view.View.requestLayout(View.java:23093)
        at androidx.constraintlayout.widget.ConstraintLayout.requestLayout(ConstraintLayout.java:3593)
        at android.view.View.requestLayout(View.java:23093)
        at android.widget.TextView.checkForRelayout(TextView.java:8908)
        at android.widget.TextView.setText(TextView.java:5730)
        at android.widget.TextView.setText(TextView.java:5571)
        at android.widget.TextView.setText(TextView.java:5528)
        at com.awkitsune.multithreadingdemonstration.MainActivity.onCreate$lambda-1$lambda-0(MainActivity.kt:23)
        at com.awkitsune.multithreadingdemonstration.MainActivity.$r8$lambda$SukcXX-DJVSL41ICI7XYhELBaGI(Unknown Source:0)
        at com.awkitsune.multithreadingdemonstration.MainActivity$$ExternalSyntheticLambda1.run(Unknown Source:2) <1 internal call>
```

Поскольку изменять состояние визуальных элементов, обращаться к ним мы можем только в основном потоке приложения или UI-потоке.

Для решения этой проблемы - взаимодействия во вторичных потоках с элементами графического интерфейса класс **View()** определяет метод **post()**:

```
fun post(action: Runnable): Boolean{   }
```

В качестве параметра он принимает задачу, которую надо выполнить, и возвращает логическое значение - **true**, если задача **Runnable** успешно помещена в очередь сообщение, или **false**, если не удалось разместить в очереди

Также у класса View есть аналогичный метод **postDelayed()**:

```
fun postDelayed(action: Runnable, millisec: Long): Boolean{   }
```

Он также запускает задачу, только через определенный промежуток времени в миллисекундах, который указывается во втором параметре.

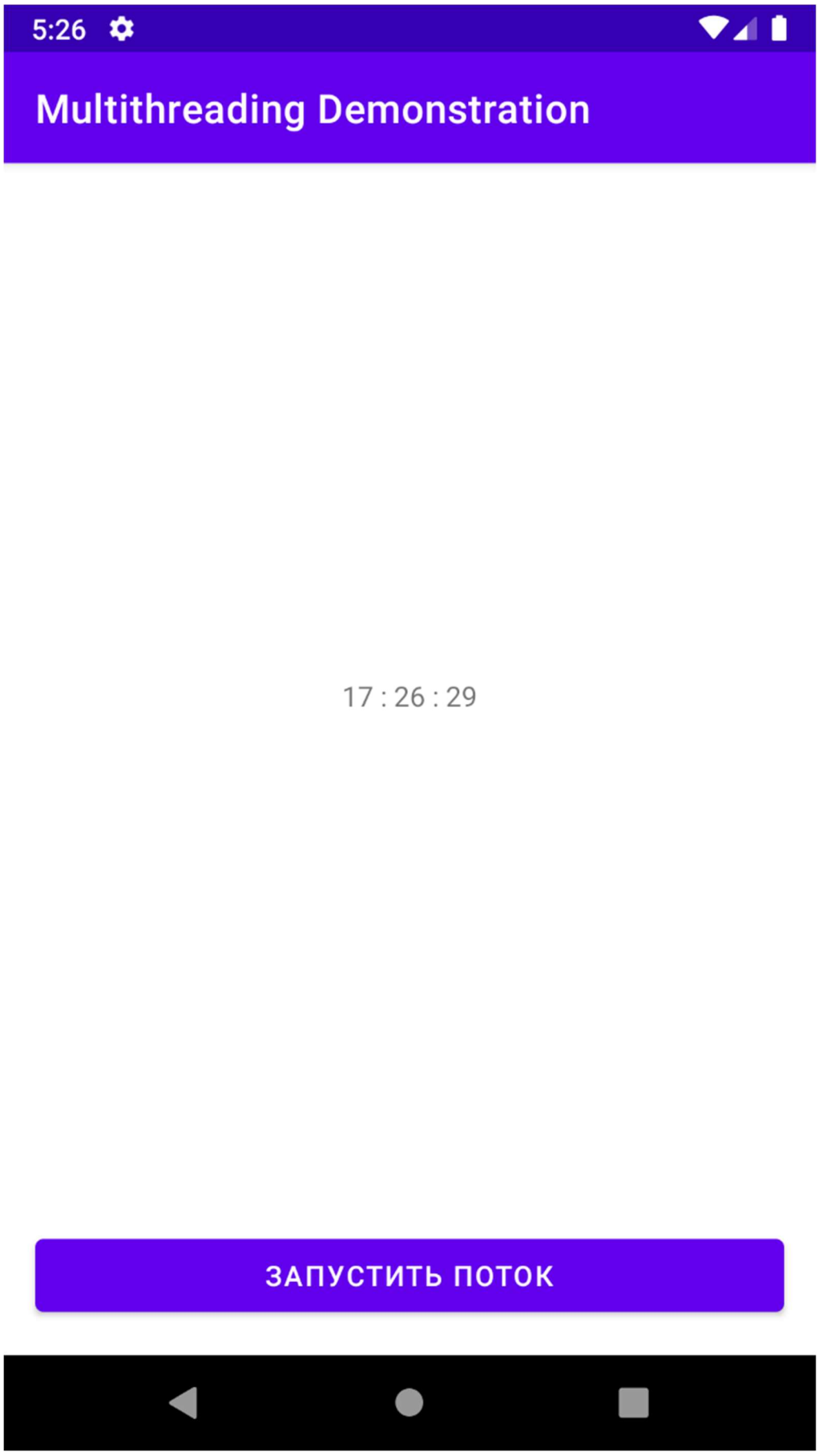
Так, изменим код **MainActivity** следующим образом

```
package com.awkitsune.multithreadingdemonstration

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.TextView
import java.util.*

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

        val textView: TextView = findViewById(R.id.textView)
        val button: Button = findViewById(R.id.button)
        button.setOnClickListener(){
            var runnable: Runnable = Runnable {
                val calendar: Calendar = Calendar.getInstance()
                val hours = calendar.get(Calendar.HOUR_OF_DAY)
                val minutes = calendar.get(Calendar.MINUTE)
                val seconds = calendar.get(Calendar.SECOND)

                textView.post(Runnable {
                    textView.text = "$hours : $minutes : $seconds"
                })
            }
            val thread: Thread = Thread(runnable)
            thread.start()
        }
    }
}
```

Теперь для обновления TextView применяется метод post:

```
textView.post(Runnable {
    textView.text = "$hours : $minutes : $seconds"
})
```

То есть здесь в методе **run()** передаемого в метод **post()** объекта Runnable мы можем обращаться к элементам визуального интерфейса и взаимодействовать с ними.



Подобным образом можно работать и с другими виджетами, которые наследуются от класса **View**.