SAMSUNG

IT Академия Kotlin

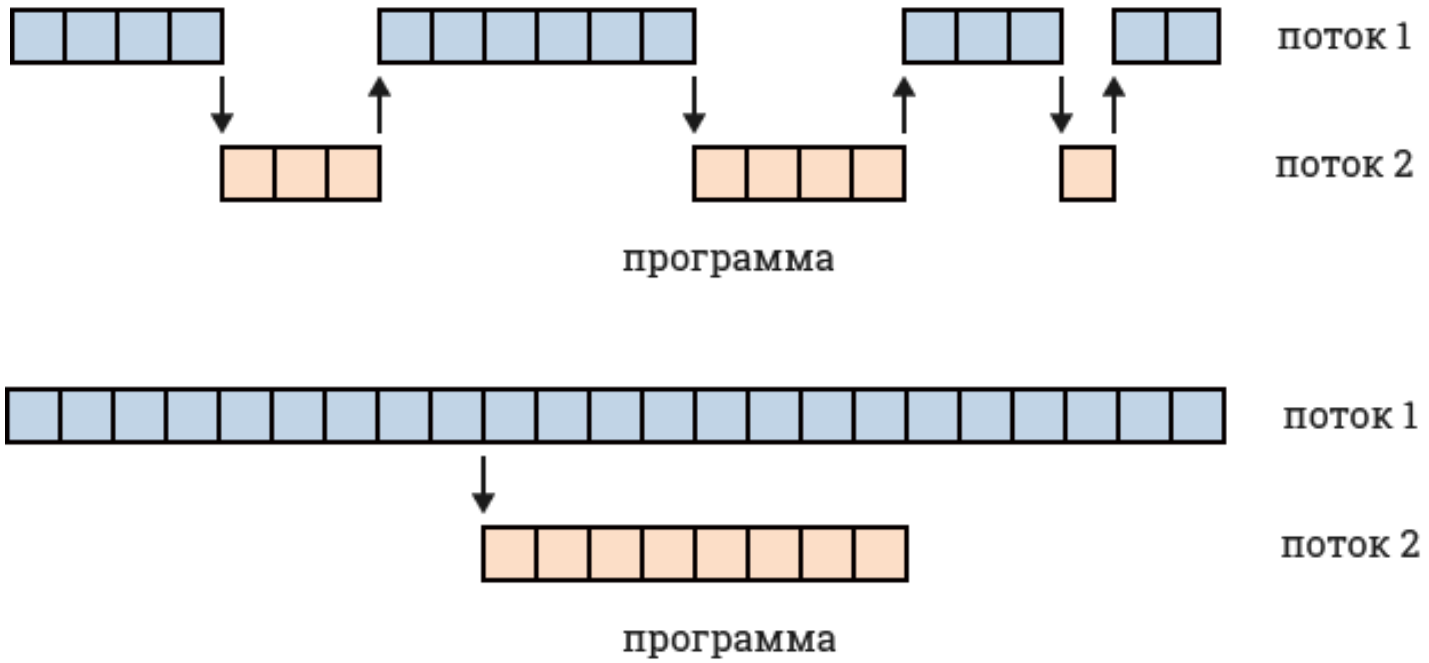
Kotlin

4.1. Многопоточность в Android приложениях

Курс по программированию от
IT Академии Samsung

4.1. Введение в многопоточность

- *Поток* можно представить как последовательность команд программы, которая претендует на использование процессора вычислительной системы для своего выполнения. Потоки одной и той же программы работают в общем адресном пространстве и, тем самым, разделяют (совместно используют) данные программы.



4.2. Потоки UI и Worker

- *Поток пользовательского интерфейса (UI-поток)* - это основной поток выполнения для приложения.
- *Рабочий поток (Worker-поток)* - это поток, в котором можно выполнять обработку, которая не должна прерывать какие-либо изменения, происходящие в UI-потоке.

```
//1
class MyThread: Thread() {
        public override fun run() {
                println("Running")
        }
}
```

```
//2
class MyRunnable: Runnable {
        public override fun run() {
println("Running")
        }
}
```

```
//1
val thread = MyThread()
thread.start()

//2
val threadWithRunnable = Thread(MyRunnable())
threadWithRunnable.start()
```

4.2. Потоки UI и Worker

SAMSUNG IT Академия Kotlin

```
public fun thread(
    start: Boolean = true,
    isDaemon: Boolean = false,
    contextClassLoader: ClassLoader? = null,
    name: String? = null,
    priority: Int = -1,
    block: () -> Unit): Thread
```

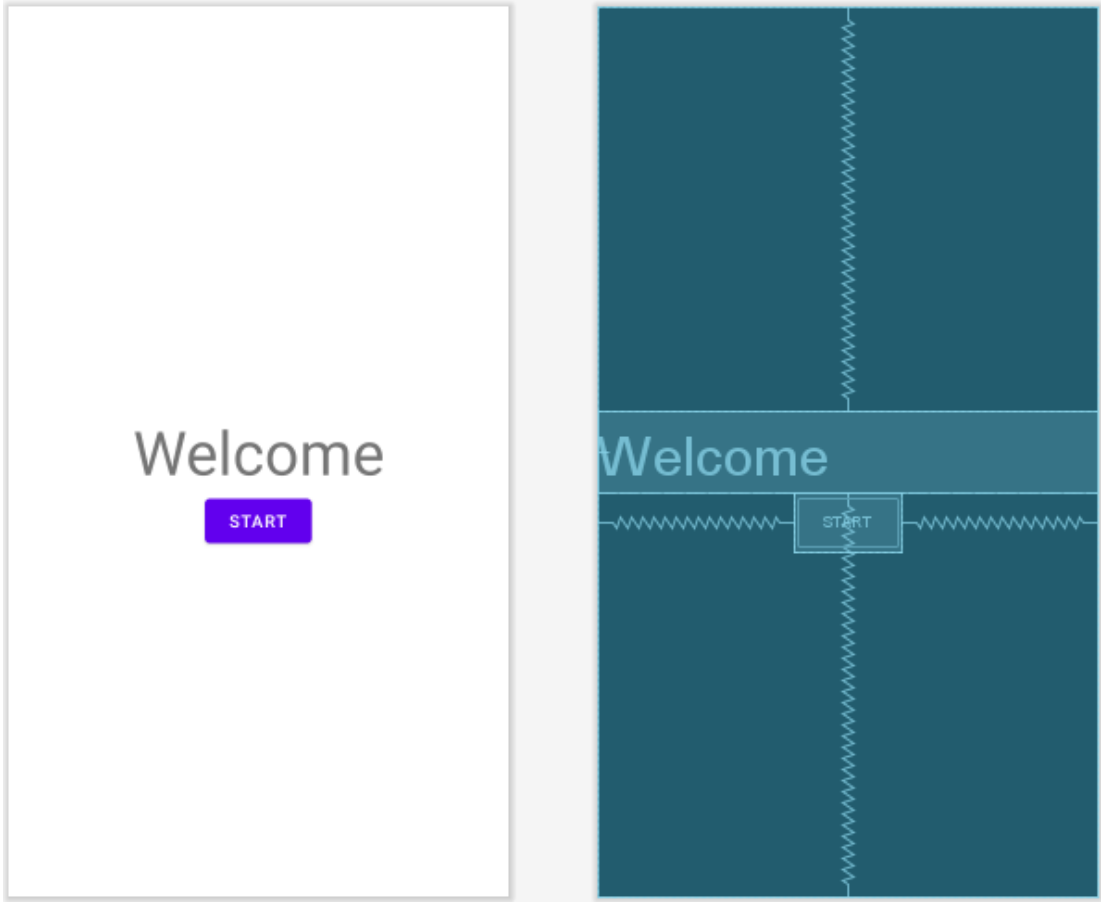
```
thread {
    Thread.sleep(1000)
    println("test")
}
```

- **start** - немедленно запустить поток;
- **isDaemon** - для создания потока как потока демона;
- **contextClassLoader** - загрузчик классов, используемый для загрузки классов и ресурсов;
- **name** - установка имени потока;
- **priority** - установка приорита потока.

4.2. Реализация UI-потока

- Если поток запущен и необходимо обновить элемент пользовательского интерфейса, можно воспользоваться функцией **runOnUIThread().**

```
val tv = findViewById<TextView>(R.id.tv1)
val btn = findViewById<Button>(R.id.btnStart)
val msg1 = "First message"
val msg2 = "Second message"
// слушатель для кнопки
btn.setOnClickListener{
   // объявляем главный поток
   Thread(Runnable {
      while (true) {
         // обновление TextView
         runOnUiThread{ tv.text = msg1 }
        // останавливаем поток на одну секунду
        Thread.sleep(1000)
        // обновление TextView
         runOnUiThread{ tv.text = msg2 }
         //останавливаем поток на одну секунду
         Thread.sleep(1000)
      }
   }).start()
}
```



4.3. Синхронизация потоков

- *Монитор* — это специальный механизм, обеспечивающий управление взаимодействием процессов и их состоянием.
- В отличие от Java, Kotlin не имеет ключевого слова **synchronized**. Следовательно, для синхронизации нескольких фоновых потоков используется аннотация **@Synchronized** или встроенная функция стандартной библиотеки **synchronized()**.

```
// синхронизированная
@Synchronized fun myFunction() { }

fun myOtherFunction() {
    // синхронизированный блок
    synchronized(this) { }
}
```

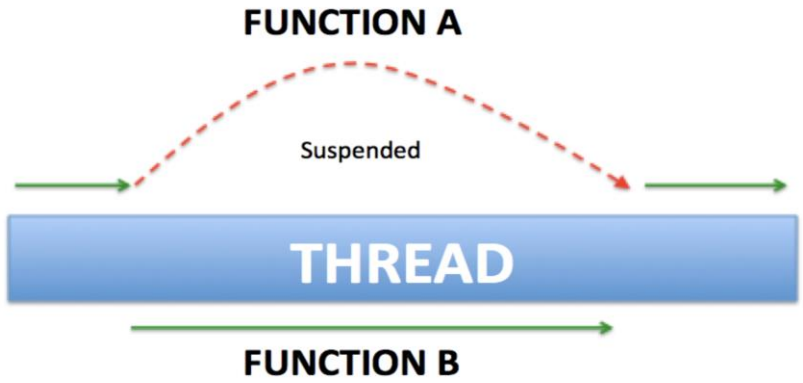
4.4. Корутины в Kotlin

Корутины — это новый способ написания асинхронного, неблокирующего кода. Корутины можно представить как облегчённый поток. Так же как и потоки, корутины могут работать параллельно, взаимодействовать между собой, ожидать друг друга.

```
dependencies {
   implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core:x.x.x"
   implementation "org.jetbrains.kotlinx:kotlinx-coroutines-
android:x.x.x"
}
```

4.4. Корутины в Kotlin

```
suspend fun doSomething(foo: Foo): Bar {
    ...
}
```



Функции остановки не могут быть вызваны из обычной функции, поэтому для них предусмотрено несколько специальных функции запуска сопрограммы, которые позволяют вызывать функцию остановки из обычной области, не требующей приостановки:

- **runBlocking**: запускает новую сопрограмму и блокирует текущий поток до его завершения,
- **launch**: запускает новую сопрограмму и возвращает ссылку на нее как на объект класса Job,
- **async**: запускает новую сопрограмму и возвращает ссылку на нее как объект Deferred <T>. Он должен использоваться вместе с функцией await, которая ожидает результата, не блокируя поток.

4.4. Пример

```
class SimpleCoroutinesActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
    //init stuff

        fab.setOnClickListener {
            launch(UI) {
                setTextAfterDelay(2, "Hello from a coroutine!")
            }
        }
    }

    private suspend fun setTextAfterDelay(seconds: Long, text: String) {
        delay(seconds, TimeUnit.SECONDS) textView.text = text
    }
}
```

4.4. Еще примеры

```
// сопрограммы запустятся немедленно и будут работать параллельно
val jobForLength1 = async {
   fetchWebsiteContents("https://myitacademy.ru/partners/").length
}
val jobForLength2 = async {
   fetchWebsiteContents("https://myitacademy.ru/news/").length
}
```

```
// сумма двух длин, используя новую сопрограмму
launch(UI) {
   val sum = jobForLength1.await() + jobForLength2.await()
   myTextView.text = "Downloaded $sum bytes!"
}
```

```
// движение виджета в цикле
launch(UI) {
   while(myTextView.x < 800) {
      myTextView.x += 10
      delay(400)
   }
}
```

Упражнение 4.1

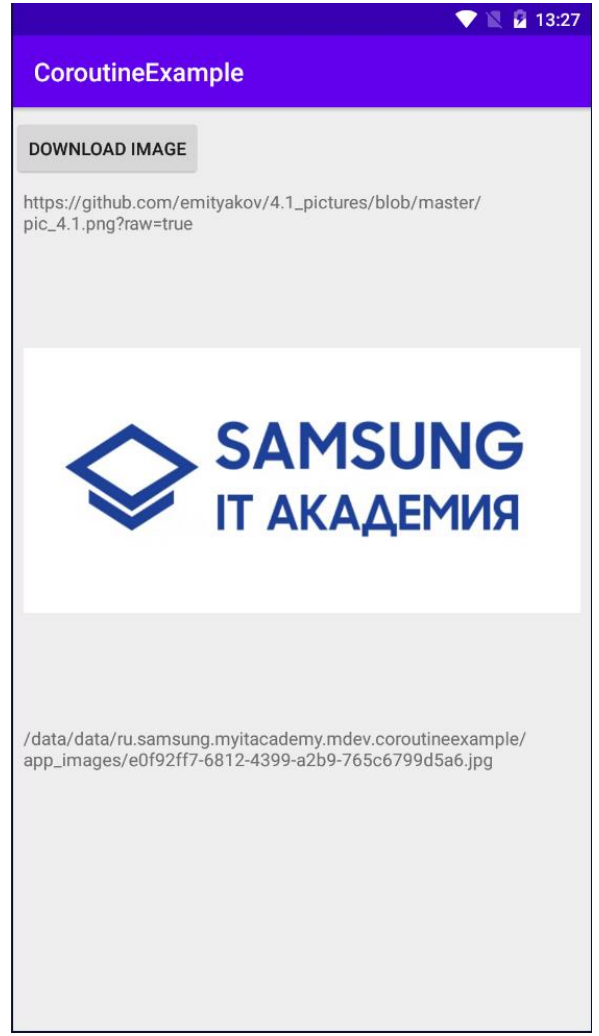
Разработаем приложение для загрузки картинки из интернета использованием корутин. На экране приложения будет отображаться кнопка "Download Image", виджет ProgressBar для отображения процесса загрузки, ImageView для отображения картинки, а также два TextView для отображения URL и URI картинки.

Для использования корутин в файл gradle необходимо добавить зависимости:

```
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.3.7'
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.7'
```

Так как в приложении будет происходить загрузка картинки из Интернета, в файл манифеста проекта необходимо добавить строчку:

```
<uses-permission android:name="android.permission.INTERNET"/>
```



SAMSUNG

IT Академия Kotlin

Kotlin

Спасибо за внимание!

Курс по программированию от
IT Академии Samsung