

Министерство науки и высшего образования Российской Федерации

Калужский филиал
федерального государственного бюджетного образовательного
учреждения высшего образования
**«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»**
(КФ МГТУ им. Н.Э. Баумана)

Ю.С. Белов, С.С. Гришунов

СЕРВИСЫ

Методические указания к выполнению лабораторной работы
по курсу «Разработка мобильного программного обеспечения»

Калуга – 2019


УДК 004.42
ББК 32.972.13
Б435

Методические указания составлены в соответствии с учебным планом КФ МГТУ им. Н.Э. Баумана по направлению подготовки 09.03.04 «Программная инженерия» кафедры «Программного обеспечения ЭВМ, информационных технологий».

Методические указания рассмотрены и одобрены:

- Кафедрой «Программного обеспечения ЭВМ, информационных технологий» (ИУ4-КФ) протокол № 51.4/9 от « 10 » апреля 2019 г.

И.о. зав. кафедрой ИУ4-КФ

 к.т.н., доцент Ю.Е. Гагарин

- Методической комиссией факультета ИУ-КФ протокол № 11 от « 19 » апреля 2019 г.

Председатель методической
комиссии факультета ИУ-КФ

 к.т.н., доцент М.Ю. Адкин

- Методической комиссией

КФ МГТУ им.Н.Э. Баумана протокол № 7 от « 7 » 05 2019 г.

Председатель методической комиссии
КФ МГТУ им.Н.Э. Баумана

 д.э.н., профессор О.Л. Перерва

Рецензент:

к.т.н., доцент кафедры ИУ6-КФ

 А.Б. Лачихина

Авторы

к.ф.-м.н., доцент кафедры ИУ4-КФ
асс. кафедры ИУ4-КФ

 Ю.С. Белов
 С.С. Гришунов

Аннотация

Методические указания по выполнению лабораторной работы по курсу «Разработка мобильного программного обеспечения» содержат описание основных этапов создания android-сервиса.

Предназначены для студентов 3-го курса бакалавриата КФ МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	4
ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ.....	5
СЕРВИСЫ	6
СОЗДАНИЕ СЕРВИСА	8
ЗАПУСК И ОСТАНОВКА СЕРВИСОВ	13
ПРИМЕР СОЗДАНИЯ СЛУЖБЫ.....	14
ТРЕБОВАНИЯ К РЕАЛИЗАЦИИ.....	17
ВАРИАНТЫ ЗАДАНИЙ.....	17
КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ	20
ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ	20
ОСНОВНАЯ ЛИТЕРАТУРА.....	21
ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА	21

ВВЕДЕНИЕ

Настоящие методические указания составлены в соответствии с программой проведения лабораторных работ по курсу «Разработка мобильного программного обеспечения» на кафедре «Программное обеспечение ЭВМ, информационные технологии и прикладная математика» факультета фундаментальных наук Калужского филиала МГТУ им. Н.Э. Баумана.

Методические указания, ориентированные на студентов 3-го курса направления подготовки 09.03.04 «Программная инженерия», содержат описание основных этапов создания android-сервиса и задание на выполнение лабораторной работы.

Методические указания составлены для ознакомления студентов с процессом создания android-сервисов. Для выполнения лабораторной работы студенту необходимы минимальные знания по программированию на высокоуровневом языке программирования Kotlin.

Требования к программному обеспечению:

- ОС Windows 7/8/10 / Ubuntu 16.04 (или старше)
- Android Studio

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ

Целью выполнения лабораторной работы является формирование практических навыков создания различных android-служб.

Основными задачами выполнения лабораторной работы являются:

1. Научиться создавать различные службы для мобильного устройства.
2. Уметь понимать схемы взаимодействия службы с другими элементами платформы Android.
3. Разработать эффективные приложения с учетом аппаратных ограничений мобильных устройств.

Результатами работы являются:

- Разработанный сервис согласно варианту задания
- Подготовленный отчет

СЕРВИСЫ

Service (служба) является компонентом приложения, который может выполнять продолжительные операции в фоновом режиме, и при этом не предоставляет интерфейса для пользователя приложения (GUI). Другой компонент приложения может запустить службу, и она будет продолжать работать в фоне, даже если пользователь переключит свое внимание на другое приложение (аналогично демонам в unix-системах). Дополнительно компонент может привязаться к службе для того, чтобы взаимодействовать с ней, и может даже выполнить обмен данными через межпроцессное взаимодействие. Например, служба может обрабатывать транзакции сети, проигрывать музыку, выполнять файловый ввод/вывод или взаимодействовать с провайдером контента (content provider), оцифровывать звук, и все это выполняется на заднем плане.

Служба может принять 2 формы: started (запущена) и bound (привязана).

Служба находится в состоянии "started", когда компонент приложения (такой как activity) запустил её вызовом `startService()`. Будучи запущенной, служба может работать в фоне бесконечно, даже если компонент, который запустил её, был уничтожен. Обычно запущенная служба выполняет одиночную операцию, и не возвращает результат в вызывавший её код. Например, служба может загрузить или выгрузить файл через сеть. Когда эта операция завершена, служба должна остановить саму себя.

Служба находится в состоянии "bound", когда компонент приложения привязался к ней вызовом `bindService()`. Привязанная служба предоставляет интерфейс типа клиент-сервер, который позволяет компоненту приложения взаимодействовать со службой, отправлять запросы, получать результаты, и даже делать то же самое с другими процессами через IPC. Привязанная служба работает, пока другой компонент приложения привязан к ней. Одновременно к службе может быть привязано несколько компонентов, но, когда они все отвязаны (`unbind`), служба уничтожается.

Служба может работать обоими способами. Независимо от того, в каком состоянии приложение - started, bound, или и то, и другое, любой компонент приложения может использовать службу (даже из отдельного приложения), тем же самым путем, как любой компонент может использовать activity — путем запуска его вместе с Intent. Однако, можно декларировать в файле манифеста службу как private, и тем самым заблокировать доступ к ней из других приложений.

СОЗДАНИЕ СЕРВИСА

Чтобы определить сервис, необходимо создать новый класс, расширяющий базовый класс Service. Можно воспользоваться готовым мастером создания класса для сервиса в Android Studio (рис. 1).

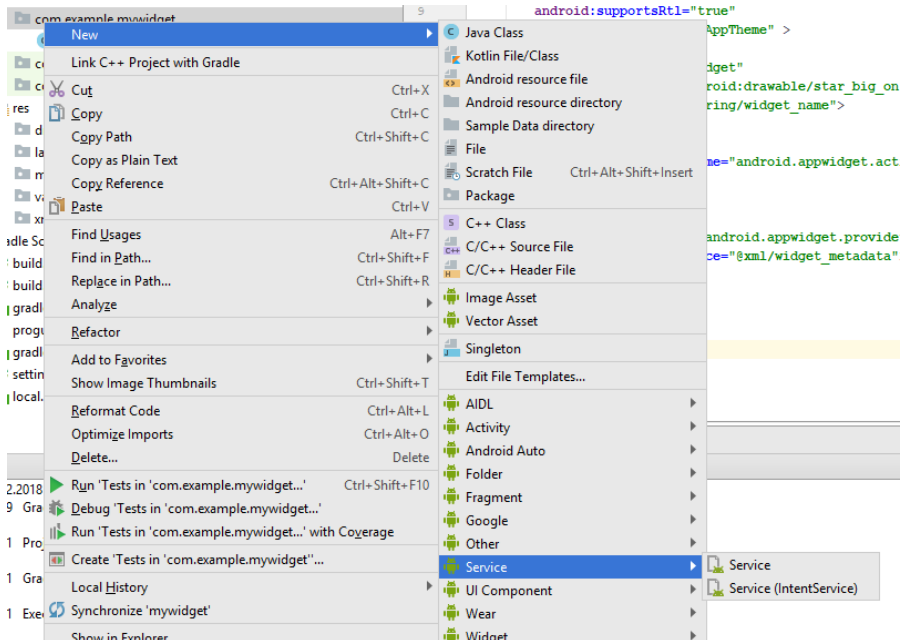


Рис. 1. Открытие мастера создания класса для сервиса в Android Studio

После выбора пункта Service появится окно (рис. 2):

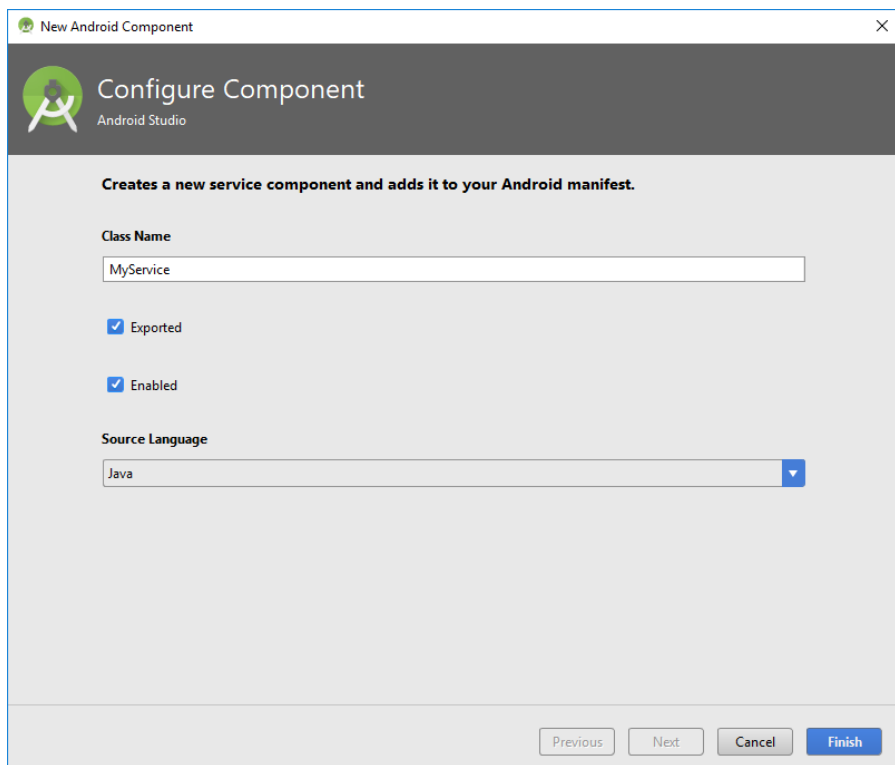


Рис. 2. Окно мастера создания класса для сервиса

Exported – предоставление возможности другим приложениям получать доступ к сервису.

Enabled – может ли сервис создаваться системой.

После создания, сервис автоматически регистрируется в манифесте в секции <application>:

```
<service
  android:name=".MyService"
  android:enabled="true"
  android:exported="true">
</service>
```

Необходимо переопределить некоторые callback-методы (методы обратного вызова), которые обрабатывают ключевые аспекты жизненного цикла службы и предоставляют (для компонентов приложения) механизм привязки службы, если это нужно.

Одни из основных callback-методов:

- **onStartCommand()**. Система вызовет этот метод, когда другой компонент, такой как activity, запросит запуск службы вызовом `startService()`. Как только этот метод завершится, служба будет запущена, и может бесконечно работать в фоне. Если Вы реализовали этот метод, то в Вашей ответственности остановить службу, когда она завершит работу, путем вызова `stopSelf()` или `stopService()` (если Вы решили использовать привязку - binding, то Вам не нужно реализовывать этот метод).
- **onBind()**. Система вызовет этот метод, когда другой компонент захочет сделать привязку к службе (например, для выполнения RPC) вызовом `bindService()`. В Вашей реализации этого метода Вы должны предоставить интерфейс, через который клиенты будут общаться со службой, путем возврата `IBinder`. Вы должны всегда реализовывать этот метод, но если Вам не нужно разрешать привязку, то этот метод должен возвращать `null`.
- **onCreate()**. Система вызовет этот метод, когда служба создана в первый раз, чтобы выполнить однократные процедуры установки (перед тем, как система вызовет либо `onStartCommand()` либо `onBind()`). Если служба уже запущена, то этот метод вызван не будет.
- **onDestroy()**. Система вызовет этот метод, когда служба больше не используется и будет уничтожена. Ваша служба должна реализовать этот метод для выполнения очистки любых используемых ресурсов, таких как потоки, зарегистрированное прослушивание событий (`registered listeners`), приемники (`receivers`) и т. п. Это последний вызов, который принимает служба.

Если компонент запускает службу вызовом `startService()`, (который в результате приведет к вызову `onStartCommand()`), то служба будет работать, пока не остановит сама себя методом `stopSelf()`, или пока другой компонент не остановит её вызовом `stopService()`.

Сервисы запускаются в главном потоке приложения; это значит, что любые операции, выполняющиеся в обработчике `onStartCommand()`, будут работать в контексте главного потока GUI. На практике при реализации сервиса в методе `onStartCommand()` создают и запускают новый поток, чтобы выполнять операции в фоновом режиме и останавливать сервис, когда работа завершена.

Такой подход позволяет методу `onStartCommand()` быстро завершить работу и даёт возможность контролировать поведение сервиса при его повторном запуске, используя одну из констант:

- `START_STICKY` – описывает стандартное поведение. Похоже на то, как был реализован метод `onStart()` в Android 2.0. Если вы вернете это значение, обработчик `onStartCommand()` будет вызываться при повторном запуске сервиса после преждевременного завершения работы. Обратите внимание, что аргумент `Intent`, передаваемый в `onStartCommand()`, получит значение `null`. Данный режим обычно используется для сервисов, которые сами обрабатывают свои состояния, явно стартуя и завершая свою работу при необходимости (с помощью методов `startService()` и `stopService()`). Это относится к сервисам, которые проигрывают музыку или выполняют другие задачи в фоновом режиме
- `START_NOT_STICKY` – этот режим используется в сервисах, которые запускаются для выполнения конкретных действий или команд. Как правило, такие сервисы используют `stopSelf()` для прекращения работы, как только команда выполнена. После преждевременного прекращения работы сервисы, работающие в данном режиме, повторно запускаются только в том случае, если получают вызовы. Если с момента завершения работы Сервиса не был запущен метод `startService()`, он остановится без вызова обработчика `onStartCommand()`. Данный режим идеально подходит для сервисов, которые обрабатывают конкретные запросы, особенно это касается регулярного выполнения

заданных действий (например, обновления или сетевые запросы). Вместо того, чтобы перезапускать сервис при нехватке ресурсов, часто более целесообразно позволить ему остановиться и повторить попытку запуска по прошествии запланированного интервала

- `START_REDELIVER_INTENT` – в некоторых случаях нужно убедиться, что команды, которые вы посылаете сервису, выполнены. Этот режим — комбинация предыдущих двух. Если система преждевременно завершила работу сервиса, он запустится повторно, но только когда будет сделан явный запрос на запуск или если процесс завершился до вызова метода `stopSelf()`. В последнем случае вызовется обработчик `onStartCommand()`, он получит первоначальное намерение, обработка которого не завершилась должным образом.

Обратите внимание, что при окончании всех операций каждый из этих режимов требует явной остановки сервиса с помощью методов `stopService()` или `stopSelf()`.

ЗАПУСК И ОСТАНОВКА СЕРВИСОВ

Чтобы запустить службу, в клиентском приложении необходимо вызывать метод `startService()`. Существует два способа вызова службы:

- явный вызов;
- неявный вызов

Пример для явного вызова службы с именем `MyService`:

```
startService(new Intent(this, MyService.class))
```

Также можно явно определить службу, создав экземпляр класса этой службы.

Пример неявного вызова службы:

```
startService(new Intent(MyService.SERVICE_ACTION))
```

Как только сервис завершил выполнение тех действий, для которых он запускался, вы должны вызвать метод `stopSelf()` либо без передачи параметра, чтобы ускорить остановку работы, либо передав значение `startId`, чтобы убедиться, что задачи выполнены для всех экземпляров, запущенных с помощью вызова `startService()`, как показано в следующем фрагменте:

```
stopSelf(startId)
```

Для остановки работы используйте метод **`stopService()`**, передавая ему объект **`Intent`**, определяющий нужный сервис. Если метод **`startService()`** вызывается для сервиса, который уже работает, обработчик **`onStartCommand()`**, принадлежащий объекту **`Service`**, будет вызван повторно. Вызовы **`startService()`** не накапливаются, поэтому единственный вызов метода **`stopService()`** завершит работу сервиса, неважно, сколько раз производился вызов **`startService()`**.

ПРИМЕР СОЗДАНИЯ СЛУЖБЫ

Создадим [службу](#), которая будет запускать на воспроизведение музыкальный файл, который будет проигрываться в фоновом режиме. Управлять службой можно будет из активности. Создайте новый проект. Для службы создайте отдельный класс PlayService. Служба будет загружать музыкальный файл sample.mp3 из каталога res/raw/ (разместите там свой MP3-файл).

Файл PlayService.kt

```
import android.app.Service
import android.content.Intent
import android.os.IBinder
import android.widget.Toast
import android.media.MediaPlayer

class PlayService : Service() {

    private var mPlayer: MediaPlayer? = null

    override fun onBind(intent: Intent): IBinder? {
        return null
    }

    override fun onCreate() {
        super.onCreate()
        Toast.makeText(this, "Служба создана",
            Toast.LENGTH_SHORT).show()
        mPlayer = MediaPlayer.create(this,
            R.raw.flower_romashka)
        mPlayer!!.isLooping = false
    }

    override fun onStartCommand(intent: Intent, flags:
        Int, startId: Int): Int {
        Toast.makeText(this, "Служба запущена",
            Toast.LENGTH_SHORT).show()
        mPlayer!!.start()
        return super.onStartCommand(intent, flags,
            startId)
    }
}
```

```

    }

    override fun onDestroy() {
        super.onDestroy()
        Toast.makeText(this, "Служба остановлена",
Toast.LENGTH_SHORT).show()
        mPlayer!!.stop()
    }
}

```

В layout-файле определим кнопки «старт» и «стоп»:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/andro
id"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:gravity="center"
android:orientation="vertical">

<Button
    android:id="@+id/button_start"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Старт" />

<Button
    android:id="@+id/button_stop"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Стоп" />
</LinearLayout>

```

В классе активности в обработчиках событий кнопок будем вызывать методы [startService\(\)](#) и [stopService\(\)](#) для управления службой:

```

import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.content.Intent

```

```

import android.view.View
import android.widget.Button

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        title = "SwipeRefreshLayout"
        val btnStart = findViewById(R.id.button_start) as
Button
        val btnStop = findViewById(R.id.button_stop) as Button

        // запуск службы
        btnStart.setOnClickListener(object:
View.OnClickListener {
            override fun onClick(view: View) {
                // используем явный вызов службы
                startService(Intent(this@MainActivity,
                PlayService::class.java))
            }
        })

        // остановка службы
        btnStop.setOnClickListener(object:
View.OnClickListener {
            override fun onClick(view: View) {
                stopService(Intent(this@MainActivity,
                PlayService::class.java))
            }
        })
    }
}

```

Запущенная служба будет выполняться независимо от состояния активности несмотря на то, что эти компоненты находятся в одном [приложении](#): если её завершить, служба все равно останется работать.

Если вы хотите, чтобы сервис запускался и после перезагрузки устройства, то следует создать приёмник широковещательных сообщений и запустить в нём сервис.

```
class BootBroadcast : BroadcastReceiver() {  
    fun onReceive(context: Context, intent: Intent) {  
        context.startService(  
            Intent(context, TestService::class.java)  
        )  
    }  
}
```

Приёмник регистрируется в манифесте с именем действия **BOOT_COMPLETED**:

```
<receiver android:name=".BootBroadcast">  
    <intent-filter >  
        <action  
android:name="android.intent.action.BOOT_COMPLETED"/>  
    </intent-filter>  
</receiver>
```

ТРЕБОВАНИЯ К РЕАЛИЗАЦИИ

Программа должна быть реализована на языке высокого уровня Kotlin.

ВАРИАНТЫ ЗАДАНИЙ

1. Создать службу органайзера. Приложение должно иметь возможность добавления, изменения, удаления событий. Также необходимо предусмотреть возможность получения ближайших событий, включения и отключения службы.
2. Создать службу будильника. Приложение должно иметь возможность установки времени будильника и мелодии. Также необходимо предусмотреть возможность включения и отключения службы.

3. Создать службу проигрывателя музыкальных файлов. Приложение должно иметь возможность создания, редактирования и удаления плейлиста. Воспроизведение должно проходить в фоновом режиме, средствами службы. Также необходимо предусмотреть возможность включения и отключения службы.
4. Создать службу контроля зарядки аккумуляторной батареи. Приложение должно иметь возможность установки предельно допустимой скорости разрядки аккумуляторной батареи, получения текущего состояния заряда. Скорость разрядки измеряется в единицах процент/минута. При превышении предельно допустимой скорости разрядки аккумуляторной батареи служба должна посылать уведомление. Также необходимо предусмотреть возможность включения и отключения службы.
5. Создать службу контроля запущенных приложений. Используя средства службы, приложение должно выводить список запущенных процессов, их общее количество. Также необходимо предусмотреть возможность включения и отключения службы.
6. Создать службу наблюдения за устройствами передачи данных. Приложение должно иметь возможность выбора устройств(а) (Wi-Fi, Bluetooth, GPS) и установки таймера для уведомления пользователя о факте работы устройства. Также необходимо предусмотреть возможность включения и отключения службы.
7. Создать службу контроля подсветки экрана устройства. Приложение должно иметь возможность установки интенсивности подсветки и установки таймера, по событию которого интенсивность подсветки должна измениться. Также необходимо предусмотреть возможность включения и отключения службы.
8. Создать службу хронографа. Приложение должно иметь возможность запуска до 5 секундомеров, выборочной их остановки, возобновления и сброса. Также необходимо предусмотреть возможность включения и отключения службы.

9. Создать службу контроля ОЗУ устройства. Используя средства службы, приложение должно выводить: общее количество памяти в килобайтах, объем доступной памяти для приложений, объем памяти, используемый для буферов, которые занимаются записью данных на диск, объем памяти в активном использовании. Также необходимо предусмотреть возможность включения и отключения службы.
10. Создать службу-планировщик браузера. Приложение должно иметь возможность ввода URL, который откроет браузер, а также возможность установки времени, когда браузер будет запущен. Также необходимо предусмотреть возможность включения и отключения службы.

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Дайте определение термину «сервис».
2. Перечислите формы, которые может принимать служба.
3. Перечислите основные callback-методы.
4. Опишите основные этапы создания сервиса.
5. Приведите метод запуска службы.
6. Приведите метод остановки службы.
7. Опишите возможности мастера создания класса для сервиса в Android Studio.
8. Дайте характеристику методу onStartCommand().
9. Приведите методы вызова служб.

ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

На выполнение лабораторной работы отводится 1 занятия (2 академических часа: 1 час на выполнение и сдачу лабораторной работы и 1 час на подготовку отчета).

Номер варианта студенту выдается преподавателем.

Отчет на защиту предоставляется в печатном виде.

Структура отчета (на отдельном листе(-ах)): титульный лист, формулировка задания (вариант), листинг (xml код разметки экрана, графическое представление, соответствующее разметке экрана, код программы и при необходимости наличие кода дополнительных классов), результаты выполнения работы (графические изображения примеров работы приложения), выводы).

ОСНОВНАЯ ЛИТЕРАТУРА

1. Семакова, А. Введение в разработку приложений для смартфонов на ОС Android / А. Семакова. - 2-е изд., испр. - М. : Национальный Открытый Университет «ИНТУИТ», 2016. - 103 с. : ил. ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=429181>
2. Введение в разработку приложений для ОС Android / Ю.В. Березовская, О.А. Юфрякова, В.Г. Вологодина и др. - 2-е изд., испр. - М. : Национальный Открытый Университет «ИНТУИТ», 2016. - 434 с. : ил. - Библиогр. в кн. ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=428937>
3. Разработка приложений для смартфонов на ОС Android / Е.А. Латухина, О.А. Юфрякова, Ю.В. Березовская, К.А. Носов. - 2-е изд., исправ. - М. : Национальный Открытый Университет «ИНТУИТ», 2016. - 252 с. : ил. - Библиогр. в кн. ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=428807>

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

4. Дэвид, Х. Разработка приложений Java EE 6 в NetBeans 7. [Электронный ресурс] — Электрон. дан. — М. : ДМК Пресс, 2013. — 330 с. — Режим доступа: <http://e.lanbook.com/book/58693> — Загл. с экрана.
5. Сильвен, Р. Android NDK. Разработка приложений под Android на C/C++. [Электронный ресурс] — Электрон. дан. — М. : ДМК Пресс, 2012. — 496 с. — Режим доступа: <http://e.lanbook.com/book/9126> — Загл. с экрана.
6. Ретабоуил, С. Android NDK: руководство для начинающих. [Электронный ресурс] — Электрон. дан. — М. : ДМК Пресс, 2016. — 518 с. — Режим доступа: <http://e.lanbook.com/book/82810> — Загл. с экрана.

7. Соколова, В.В. Разработка мобильных приложений: учебное пособие / В.В. Соколова ; Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Томский государственный университет», Министерство образования и науки Российской Федерации. - Томск: Издательство Томского политехнического университета, 2015. - 176 с.: ил., табл., схем. - Библиогр. в кн.. - ISBN 978-5-4387-0369-3; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=442808> (15.06.2017).
8. Баженова, И.Ю. Язык программирования Java / И.Ю. Баженова. - М.: Диалог-МИФИ, 2008. - 254 с. : табл., ил. - ISBN 5-86404-091-6 ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=54745> (15.06.2017).
9. Джошуа Блох Java. Эффективное программирование [Электронный ресурс]/ Джошуа Блох— Электрон. текстовые данные. — Саратов: Профобразование, 2017.— 310 с.— Режим доступа: <http://www.iprbookshop.ru/64057.html> .— ЭБС «IPRbooks»

Электронные ресурсы:

10. Научная электронная библиотека <http://eLIBRARY.RU>
11. Электронно-библиотечная система <http://e.lanbook.com>
12. Электронно-библиотечная система «Университетская библиотека онлайн» <http://biblioclub.ru>
13. Электронно-библиотечная система IPRBook <http://www.iprbookshop.ru>
14. Базовый сайт о системе Android - https://www.android.com/intl/ru_ru
15. Разработка приложения на базе Android - <https://developer.android.com/index.html>