

Документ подписан простой электронной подписью

Информация о владельце:

ФИО: Шебухова Татьяна Александровна

Должность: Директор Пятигорского института (филиал) Северо-Кавказского
федерального университета

Дата подписания: 12.09.2023 г.

Уникальный программный ключ:

d74ce93cd40e39275c3ba2f58486412a1c8ef96f

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

Методические указания

по выполнению лабораторных работ

по дисциплине «ПРОГРАММИРОВАНИЕ МОБИЛЬНЫХ УСТРОЙСТВ»

для студентов направления подготовки /специальности

09.03.02 Информационные системы и технологии

(ЭЛЕКТРОННЫЙ ДОКУМЕНТ)

Введение

В лабораторный практикум по дисциплине «Программирование мобильных устройств» включены лабораторные работы по основным разделам этой дисциплины, читаемой на кафедре «Систем управления и информационных технологий». Лабораторные работы ориентированы на приобретение студентами навыков программирования на языке Java для мобильной платформы Android. Работы ориентированы на использование систем автоматизированного проектирования AndroidStudio.

Содержащиеся в практикуме сведения теории, методические указания и рекомендации по выполнению лабораторных работ позволяют использовать его в качестве дополнительного пособия для закрепления курса лекций.

Целью данного лабораторного практикума является поэтапное формирование

у студентов знаний, умений и навыков создания приложения для мобильных устройств, изучение языка программирования Java.

Практикум предназначен для студентов Северо-Кавказского федерального университета и может быть полезным для всех желающих ознакомиться с основами программирования мобильных устройств.

Данный вид работы играет важную роль в формировании практических навыков программирования и способствует формированию следующих образовательных компетенций:

Лабораторная работа №1. Установка AndroidStudio

До недавнего времени компания Google поддерживала две платформы разработки – AndroidStudio и Eclipse. Поскольку большинство разработчиков мобильных приложений используют среду разработки AndroidStudio и она имеет больше возможностей, от поддержки разработки в Eclipse компания Google отказалась. Таким образом, осталась одна поддерживаемая система – AndroidStudio.

Рассмотрим процесс установки и настройки окружения.

Поскольку разработка приложения для ОС Android осуществляется на языке программирования Java, необходимо скачать и установить пакет JDK

(JavaDevelopmentKit) – этот пакет содержит все необходимые библиотеки для нативной разработки на языке программирования Java. Последняя доступная версия располагается на странице компании Oracle, в соответствующем разделе (<http://www.oracle.com/technetwork/java/javase/downloads/index.html>).

Также необходимо скачать и установить пакет разработчика Android – ADT(AndroidDevelopmentTools), который можно найти на официальной странице в сети по адресу: <http://developer.android.com/intl/ru/sdk/index.html>. В последних версиях AndroidStudio уже содержит в себе пакет ADT, по- этому достаточно только загрузить и установить среду разработки под выбранную платформу. На сегодняшний день поддерживаются следующие операционные системы Linux / Windows / MacOS.

Процесс запуска установки пакета JDK представлен на рис. 1.1.

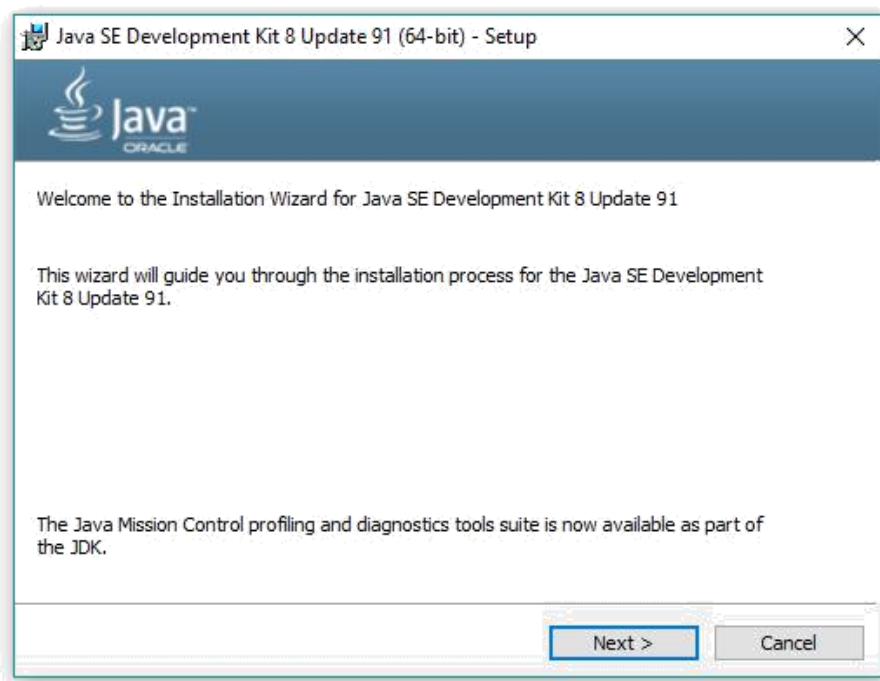


Рис. 1.1. Начало установки JDK

Следующий шаг – установка пакета JDK (рис. 1.2).

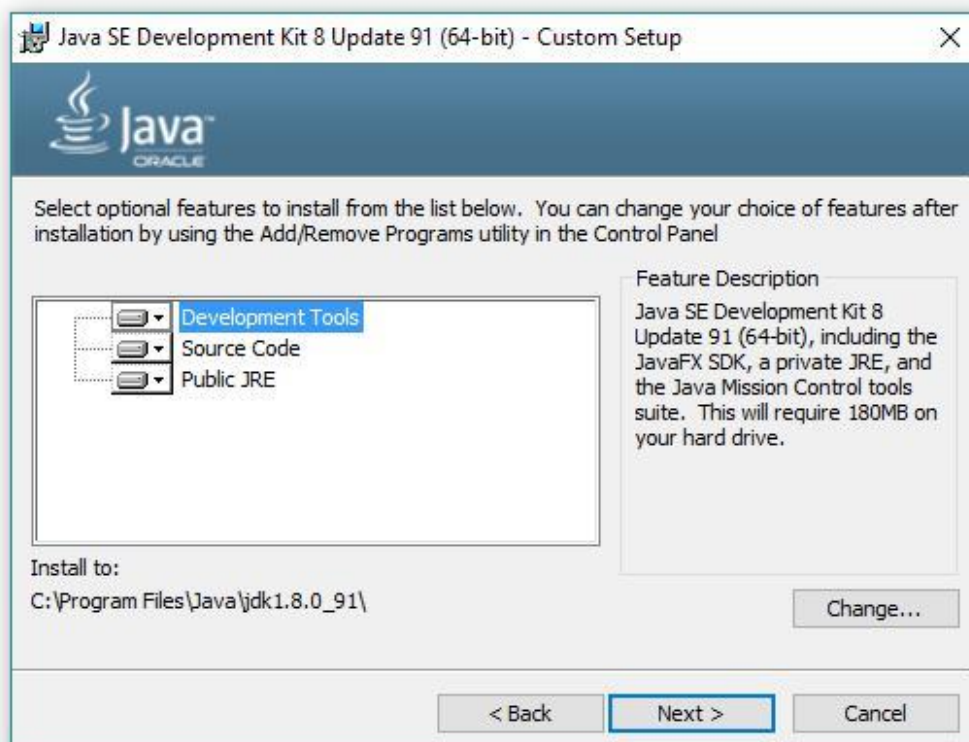


Рис. 1.2. Выбор пути для установки JDK

Следом за JDK процесс установки предложит выбрать путь для установки 15

JRE (рис. 1.3).

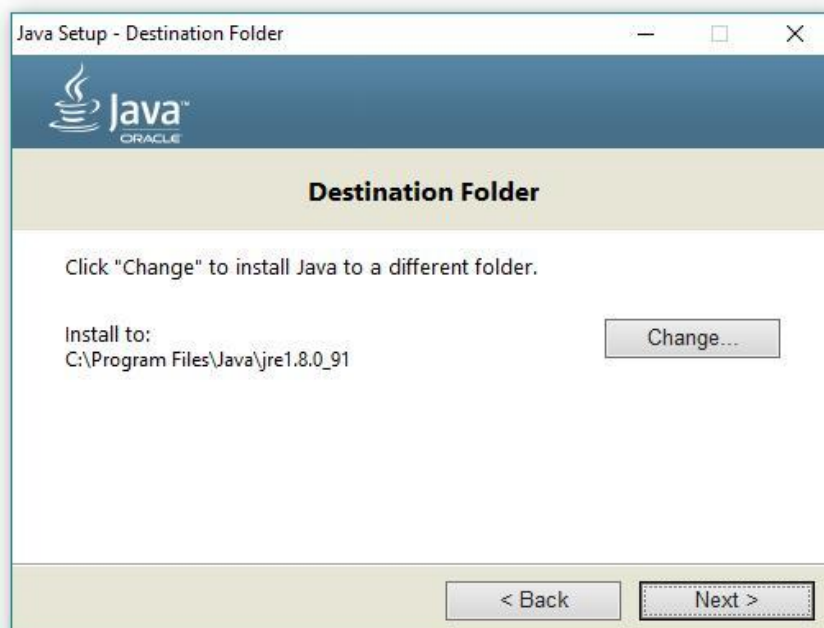


Рис. 1.3. Выбор пути для установки JRE

Следующим шагом будет установка самой AndroidStudio, скачать её можно на сайте www.developer.android.com. Запустив исполняемый файл установки и нажав Next, попадем на экран выбора компонентов для установки (рис. 1.4).

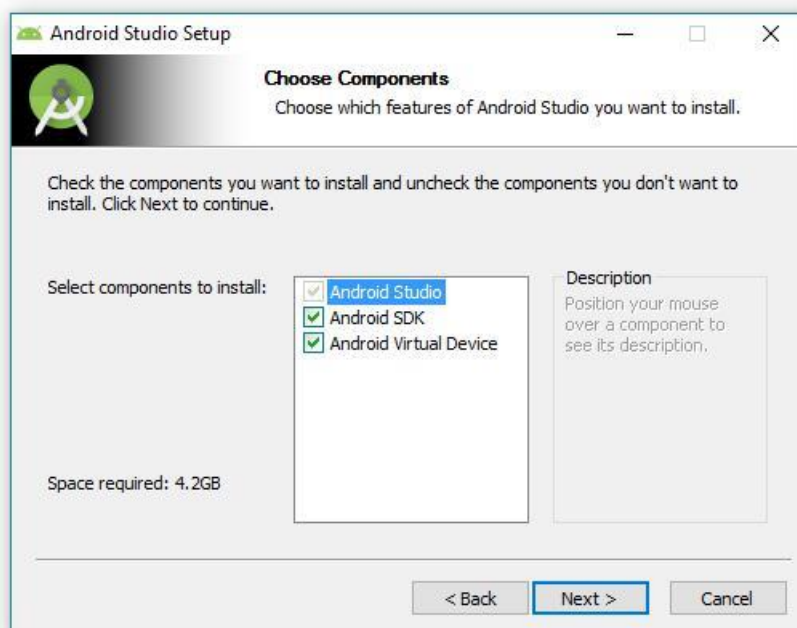


Рис. 1.4. Выбор компонентов для установки

Android Studio – среда разработки (Integrated Development Environment,

сокращенное название IDE) на основе популярной IntelliJ IDEA для разработки на языке программирования Java. AndroidSDK – среда разработки приложений для операционной системы Android, включает в себя несколько инструментальных средств, которые позволяют компилировать и отлаживать создаваемые приложения. AndroidVirtualDevice (AVD) – эмулятор мобильного устройства на ОС Android. Используя эмулятор, можно запускать и тестировать приложения без использования реального Android-устройства.

Далее, приняв лицензионное соглашение, требуется выбрать путь, куда необходимо установить AndroidStudio и SDK вместе с AVD.

Важно! В записи пути не должно быть русских символов. Если учетная запись имеет русские буквы в своем имени, следует выбрать другой путь для SDK (рис. 1.5).

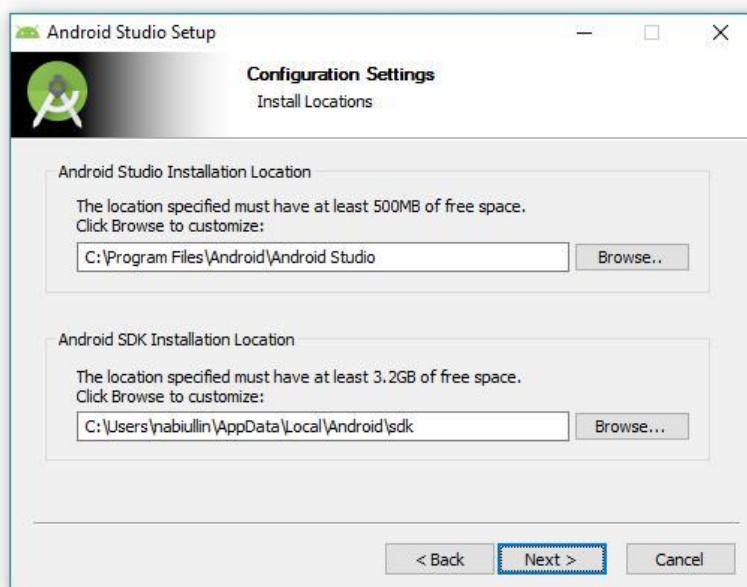


Рис. 1.5. Выбор пути для установки AndroidStudio и SDK вместе с AVD

Нажав Next, начнется процесс установки. После того как он завершится, открываем AndroidStudio и начинаем настройку окружения. В первую очередь появится окно с предложением импорта конфигурации предыдущей версии AndroidStudio (рис. 1.6).

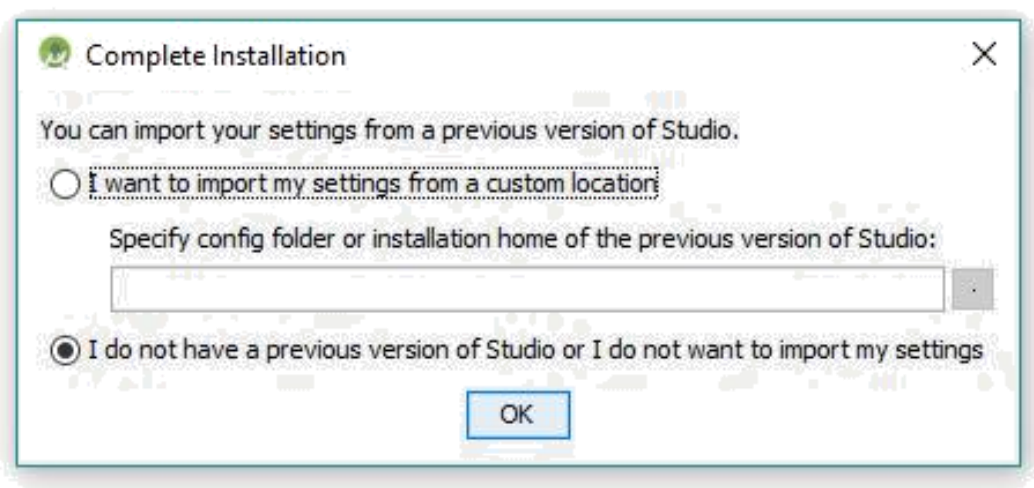


Рис. 1.6. Импорт конфигураций предыдущей версии

Выбираем второй пункт «Я не имею предыдущей версии студии или не хочу импортировать мои конфигурации». Далее откроется окно с предложением настроить AndroidStudio. Нажав Next, попадаем на окно с выбором типа настройки (рис. 1.7). Standard включает в себя стандартные настройки IDE, т. е. будут установлены все минимально необходимые компоненты для программирования. Custom позволит настроить окружение в индивидуальном порядке.

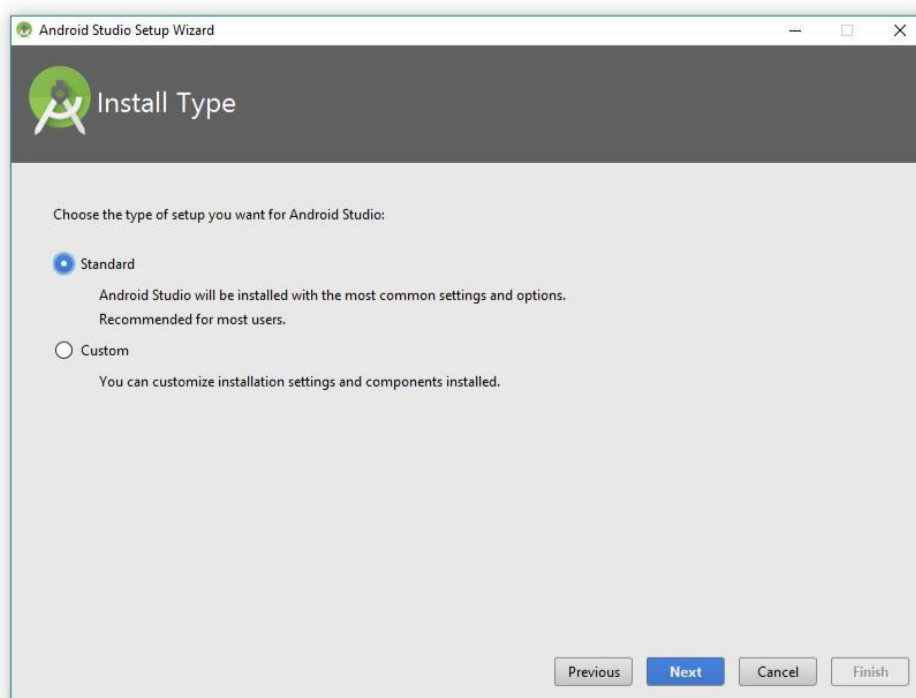


Рис. 1.7. Выбор типа настройки AndroidStudio

По завершении быстрой настройки будут загружены обновления AndroidStudio или SDK, если таковые имеются. Далее откроется начальное окно (рис. 1.8).

В данный момент AndroidSDK обладает необходимым минимумом инструментов и актуальной версией платформы для начала разработки, но если необходимо расширить возможности, можно перейти вменеджерSDK, выбрав опцию – ConfigureAndroidSDK.

Во вкладке SDKPlatforms (рис.1.9) можно выбрать доступные версии AndroidSDK.



Рис. 1.8. Начальное окно AndroidStudio

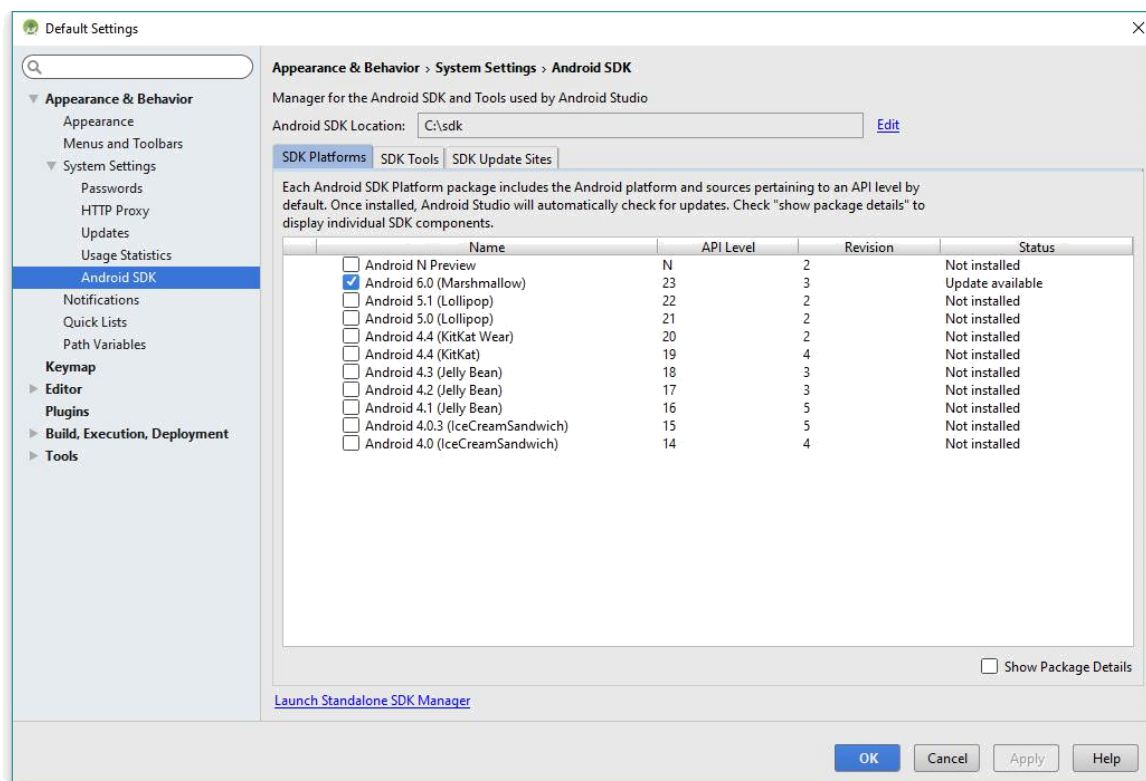


Рис. 1.9. Менеджер платформ

Во вкладке SDKTools (рис. 1.10) необходимо выбрать инструменты, которые могут понадобиться при разработке приложений. Кроме того, в большинстве случаев, если какой-либо из необходимых пакетов не был установлен, AndroidStudio укажет на необходимость установки требуемого пакета.

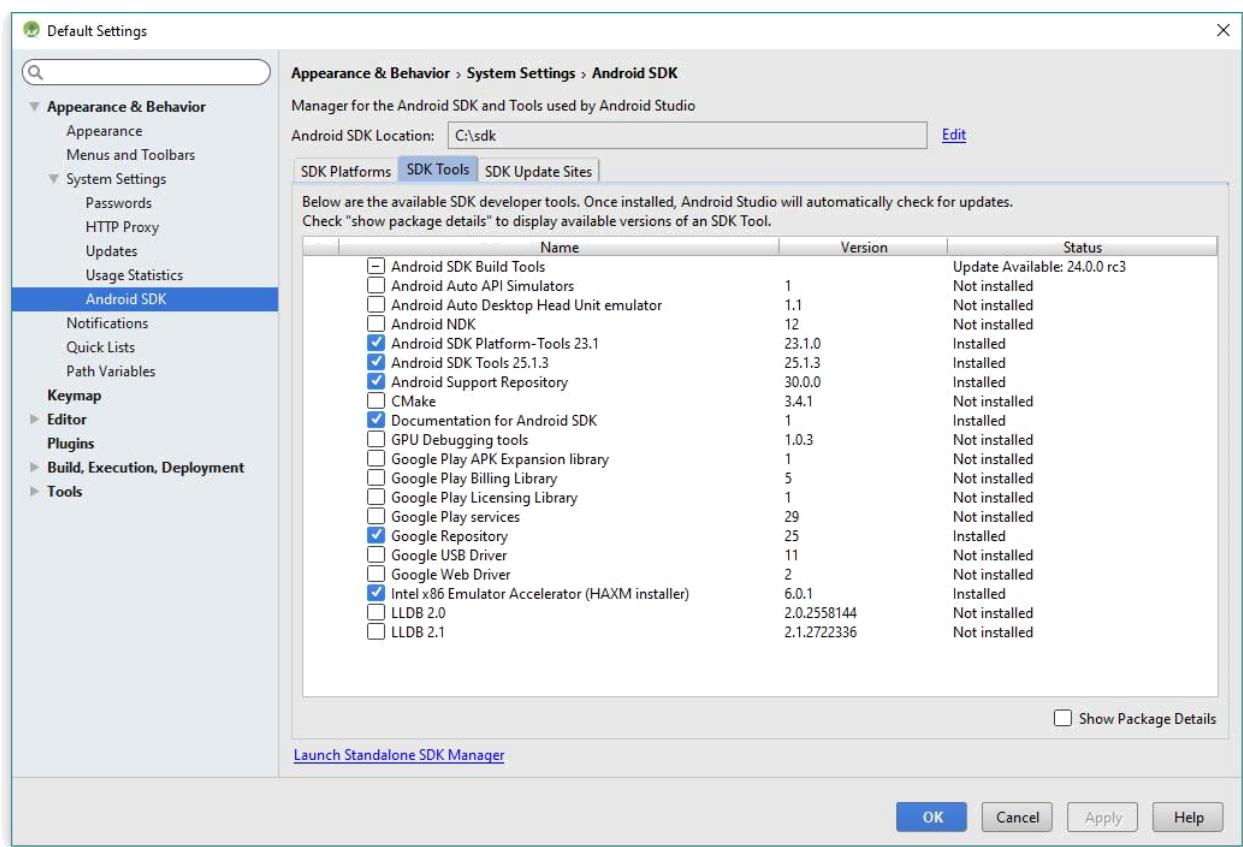


Рис. 1.10. Менеджер инструментов

Для того чтобы скачать какой-либо пакет, необходимо установить галочку напротив пункта, в котором указано название этого пакета, и подтвердить выбор нажатием кнопки ОК (рис. 1.11).

В появившемся окне отразится текст лицензионного соглашения, которое необходимо принять. Для этого выбираем пункт **Assent** (рис. 1.12) и нажимаем **Next**.

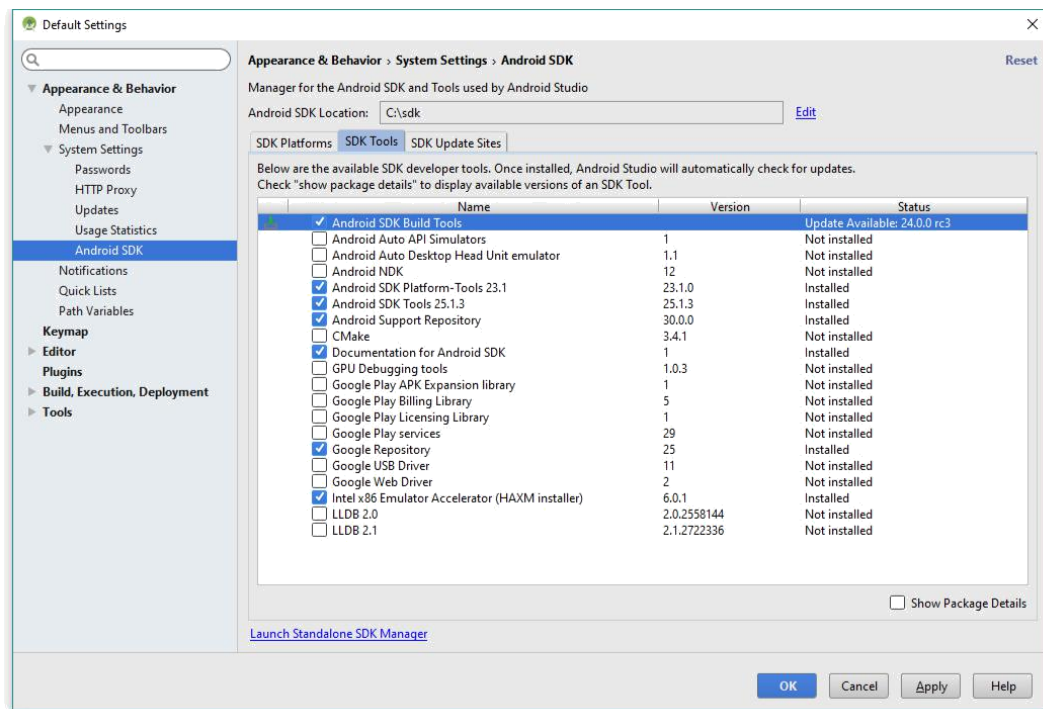


Рис. 1.11. Выбор установочных библиотек

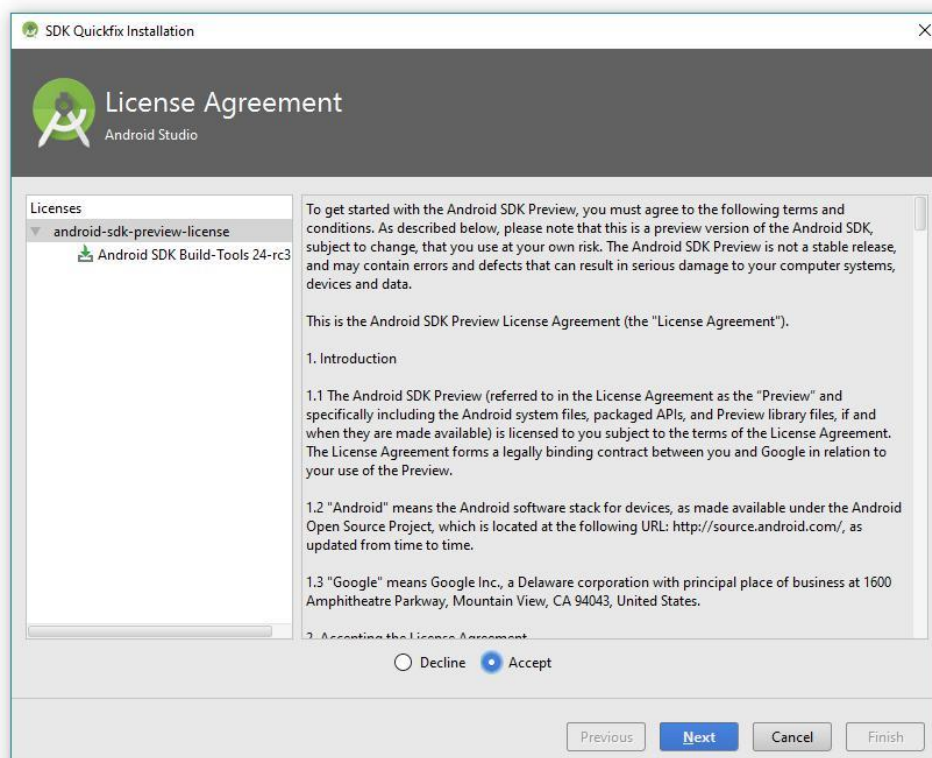


Рис. 1.12. Лицензионное соглашение

После нажатия на кнопку Next начнется скачивание, распаковка и установка необходимого вам компонента SDK.

Рабочее окружение готово для работы.

Вопросы:

1. Перечислите все версии ОСAndroid.
2. Какие средства необходимы для начала разработки под ОСAndroid?
3. Перечислите основные преимущества и недостатки ОСAndroid.
4. Под какими ОС возможно разрабатывать программное обеспечение под ОСAndroid?

Лабораторная работа 2. Создание нового проекта

Для создания проекта необходимо из контекстного меню среды разработки AndroidStudio выбрать File →NewProject. Появится диалоговое окно (рис. 2.1).

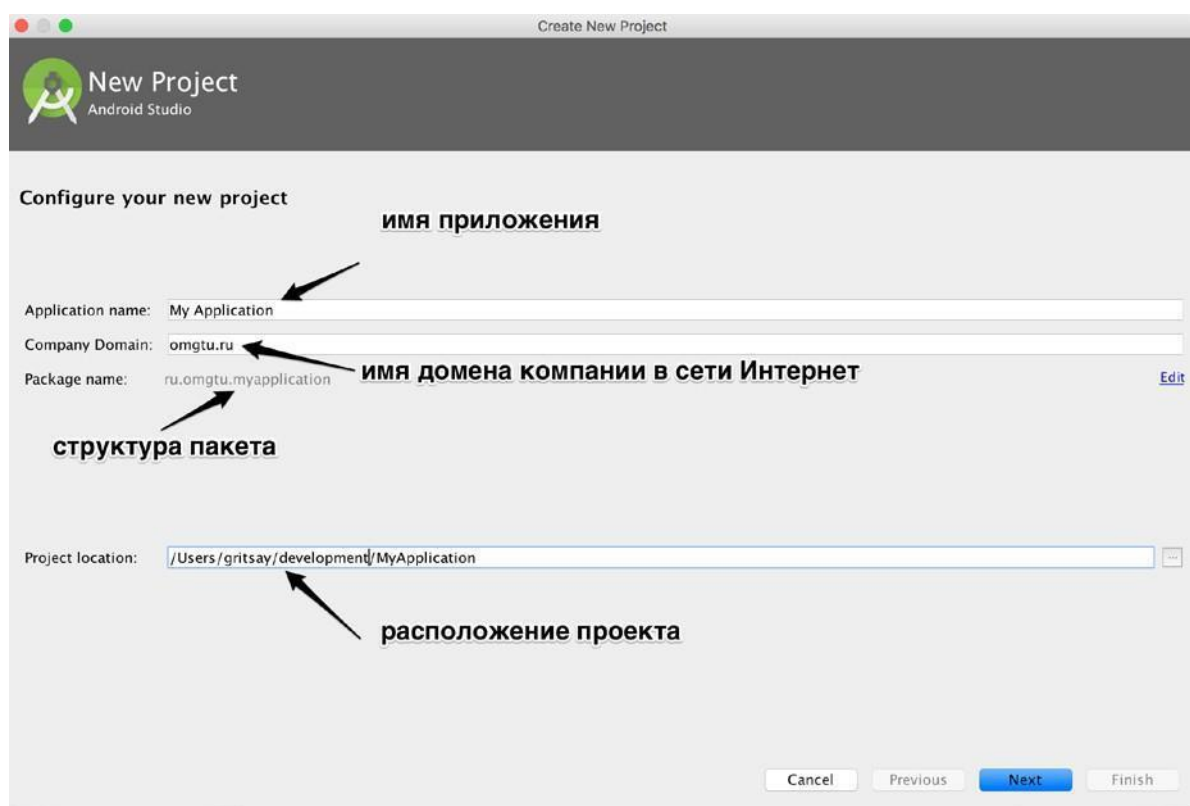


Рис. 2.1. Диалоговое окно создания нового проекта

В нем необходимо указать имя проекта, наименование идентификатора приложения – в нашем случае omgtu.ru (обычно указывается по имени сайта

организации), при этом наименование PackageName формируется автоматически – в обратной последовательности имени домена с именем проекта в конце строки. Нажимаем кнопку Next и попадаем на диалоговое окно выбора версии SDK, под которую будет собираться проект (рис. 2.2).

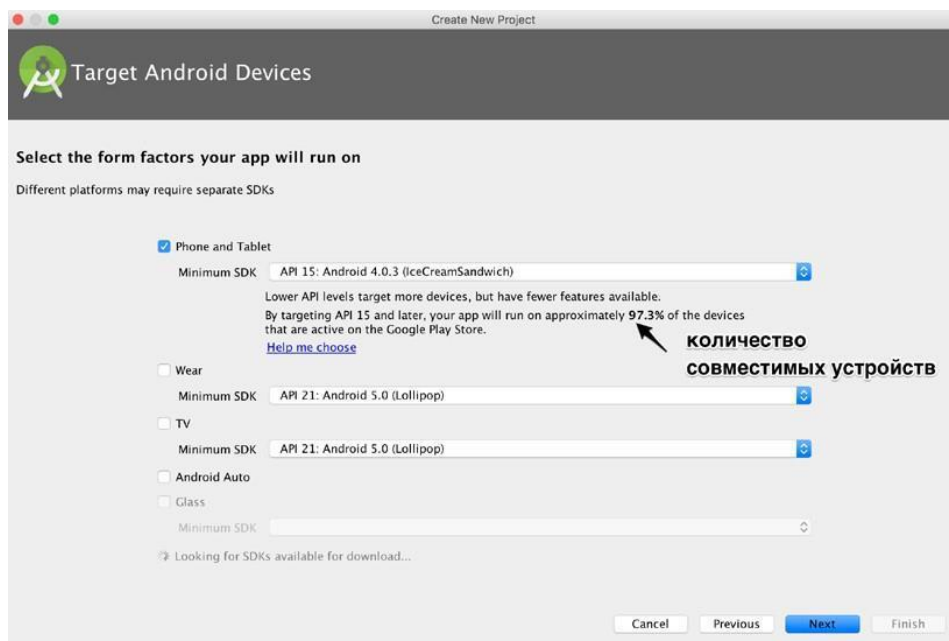


Рис. 2.2. Диалоговое окно выбора SDK для разработки

В нашем случае система сама предложила осуществить поддержку (начиная с операционной системы Android 4.0.3). Обратите внимание на строку, представленную ниже, в которой написано, что выбранная конфигурация будет совместима с 97,3 % существующих Android устройств. Также в интерактивном режиме можно посмотреть, какие еще существуют варианты, кликнув на надписи «Helpmechoose». В данном окне возможно выбрать дополнительные модули, для которых будет реализована программа Wear (платформа для программирования часов GoogleWear), TV (модуль для программирования устройств AndroidTV) и AndroidAuto (платформа Android для автомобилей).

Нажав кнопку Next, переходим на следующий экран, где представлена возможность выбора перенастроенной конфигурации элементов управления Activity (рис. 2.3).

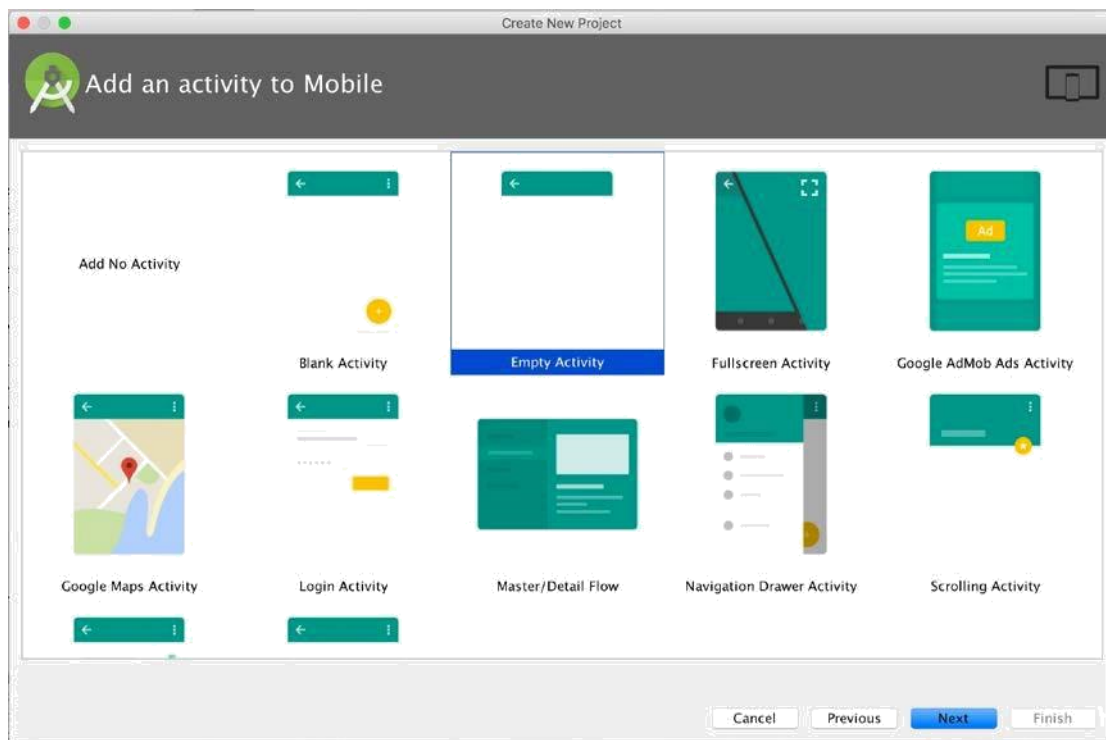


Рис. 2.3. Диалоговое окно выбора элементов Activity

На экране выбора элементов Activity необходимо указать один из подходящих вариантов с расположением элементов управления. В нашем случае элементы управления будут добавлены вручную, поэтому выбираем экранную форму без каких-либо элементов управления «EmptyActivity». Структура созданного проекта отображается в древовидном виде (рис. 2.4), что позволяет упростить навигацию по файлам.

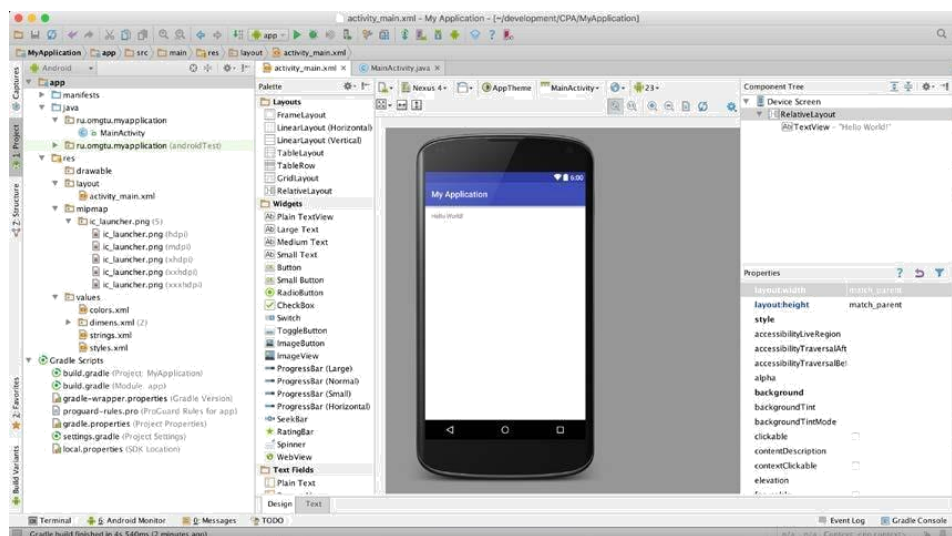


Рис. 2.4. Структура проекта в AndroidStudio

В папке GradleScripts представлены файлы конфигурации. Для проекта, созданного «по умолчанию», существует два файла конфигурации build.gradle – один включает в себя конфигурацию проекта, другой – конфигурацию модуля разрабатываемого приложения.

В папке ru.omg.tu.myapplication располагается единственный в нашем случае файл Activity, папка res содержит в себе все ресурсы приложения, настройки, константы, стили. Так в папке midmap представлено 5 иконок приложения в формате .png для разных разрешений экранов.

Чтобы проект можно было откомпилировать и запустить, в случае, если используется не прямое подключение к интернету, а через прокси-сервер, необходимо внести в файл gradle.properties следующие строки:

#для *протокола*
`httpsystemProp.http.proxyHost=hostnamesystemProp.http.proxyPort=hostport`
протокола
`httpssystemProp.https.proxyHost=hostnamesystemProp.https.proxyPort=hostport`

где hostname – имя хоста прокси сервера, hostport – номер порта, по которому доступен прокси-сервер.

В прил. А приведена структура build.gradle скрипта, который отображает команды для сборки проекта. В нашем случае данный файл был сгенерирован

автоматически при создании проекта. В секции `defaultConfig` определен идентификатор приложения (каждое приложение, находящееся в магазине приложений PlayMarket, имеет свой уникальный идентификатор), минимальная версия поддерживаемого SDK (`minSdkVersion`). Этот параметр характеризует версию операционной системы, на которой будет работать приложение. Если версия операционной системы устройства будет меньше, чем версия, определенная этим параметром, программа будет недоступна для скачивания с электронного магазина (в случае ее распространения средствами PlayMarket). Параметр `buildToolsVersion` определяет версию SDK пакета ADT, которая будет использоваться для компиляции проекта, а параметр `compileSdkVersion` определяет версию Android устройства, под которую будет собран билд. Свойства `versionCode` и `versionName` необходимы больше как информационные параметры для программиста и системы распространения PlayMarket. Они позволяют не нарушать принципы нумерования версий программ. В свою очередь версионность систем может задаваться релизной политикой, которая определена документально при разработке той или иной системы.

Секция `buildTypes` описывает специальные параметры сборки билда, его тип и опции обфускации.

Секция `dependencies` описывает дополнительные пакеты (зависимости), которые используются в проекте. Данные зависимости можно добавить как вручную в файл `gradle.build`, так и в автоматическом режиме поиском по существующим библиотекам в системе `maven` и выбором их в диалоговом интерфейсном окне (рис. 2.5).

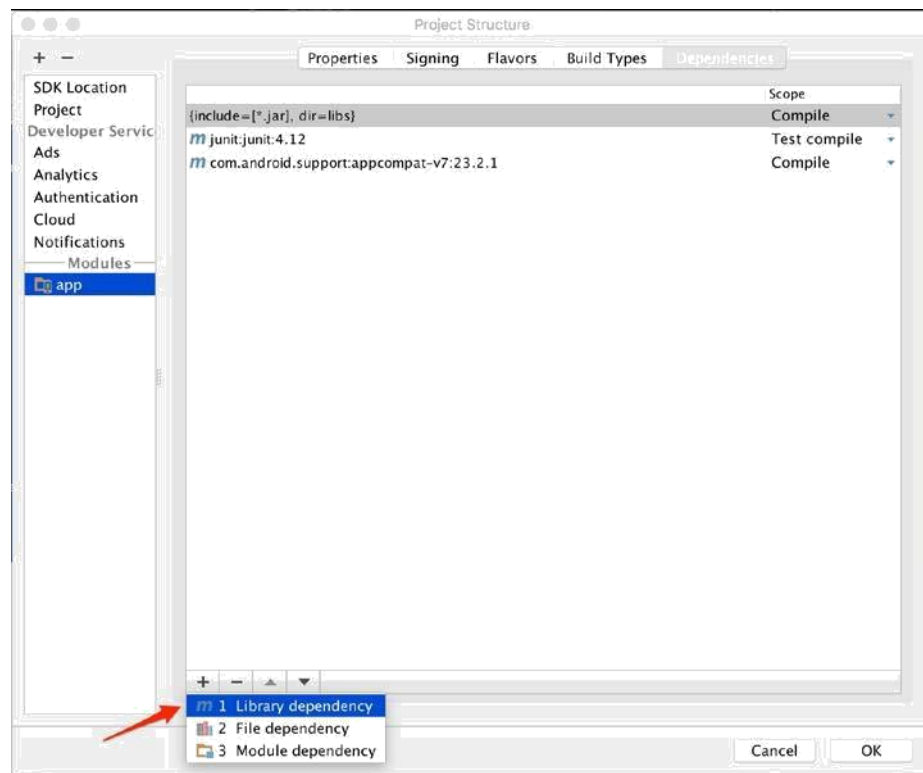


Рис. 2.5. Диалоговое окно добавления зависимостей

Для того чтобы вызвать это диалоговое окно, необходимо выполнить Build → Edit Libraries and Dependencies.

В диалоге нужно нажать «+» и выбрать Librarydependency. Разумеется, поиск будет возможен только при наличии доступа в сеть Интернет. В других случаях, например, при наличии предварительно скачанной библиотеки, можно воспользоваться опцией Filedependency или Moduledependency (зависит от типа библиотеки).

При написании программы под ОС Android существует еще один важный файл конфигурации, который называется AndroidManifest.xml. Пример такого файла приведен в прил. Б. Основное, что на сейчас следует усвоить, – это принцип осуществления входа в приложение, т. е. явное указание основного элемента Activity, который будет выполнен после запуска приложения. В примере это MainActivity, он определен как основной с использованием интент-фильтров следующих типов android.intent.action.MAIN и android.intent.category.LAUNCHER.

Таким образом, рассмотрев основные возможности проекта
для 45

операционной системы Android, мы можем его скомпилировать. Для этого необходимо выполнить Build → Build APK.

Вопросы:

1. Разработку под какие типы устройств поддерживает IDE Android Studio?
2. Перечислите все основные файлы проекта, созданного по умолчанию.
3. Каким образом возможно добавление сторонней библиотеки в разрабатываемое приложение?
4. Что такое apk-файл? Как его получить?

Лабораторная работа 3. Жизненный цикл Activity

Как мы рассматривали выше, одним из ключевых компонентов системы Android является компонент Activity. Жизненный цикл этого компонента контролируется операционной системой Android, а сам компонент взаимодействует с пользователем посредством графического интерфейса. Например, если пользователь в определенный момент времени работает с каким-либо окном в приложении, этому окну система Android присваивает наивысший приоритет, и наоборот, если пользователь перевел приложение в режим бэкграунд и не использует его, система решает, что приложение необходимо остановить, чтобы выделить дополнительные ресурсы для других активных приложений. В общем случае жизненный цикл Activity представлен на рис.3.1.

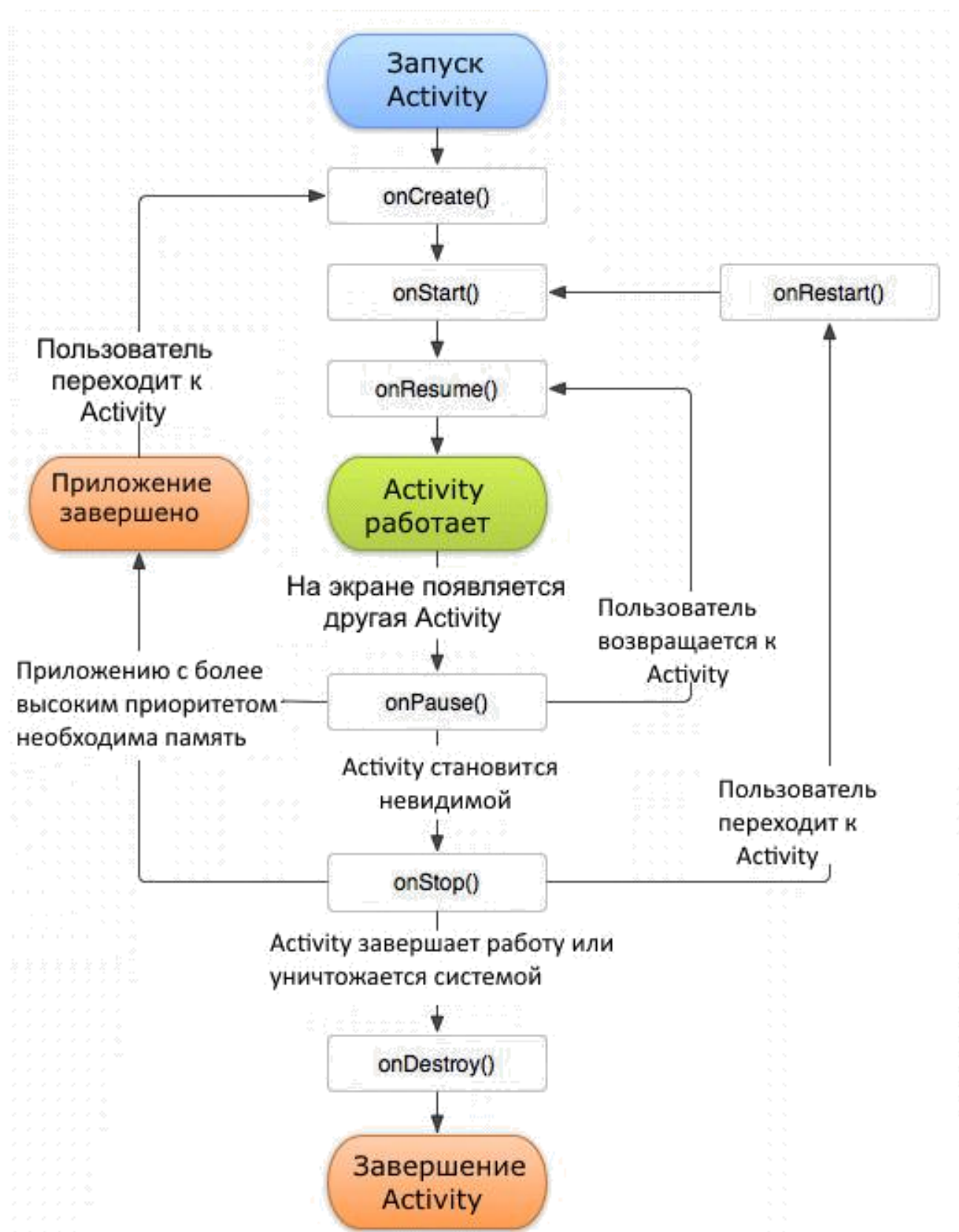


Рис. 3.1. Жизненный цикл Activity

Список основных методов, определяющих жизненный цикл Activity, следующий:

- `protected void onCreate();`
- `protected void onStart();`
- `protected void onRestart();`
- `protected void onResume();`
- `protected void onPause();`
- `protected void onStop();`
- `protected void onDestroy();`

Метод `onCreate()` всегда вызывается при создании и перезапуске `Activity` (рис. 3.2). Нотация `@Override` обозначает, что переопределяется стандартный метод класса `Activity.class`. Внутри этого метода интерфейс связывается с классом `Activity` посредством метода `setConetentView()`. Этот метод принимает в параметре объект `Bundle`, содержащий текущее состояние пользовательского интерфейса, сохраненное при последнем вызове обработчика `onSavedInstanceState`. Также инициализируются статические данные.

```
@Override
public void onCreate() {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

Рис. 3.2. Пример переопределения метода `onCreate()`

Метод `onStart()` запускается при старте `Activity` вслед за методом `onCreate()`, но не всегда, поскольку метод `onStart()` также вызывается при возобновлении работы приложения после того, как оно было остановлено. При остановке вызывается метод `onStop()`, это происходит в момент скрывания активности, например, когда открывается другое приложение.

Метод `onResume()` вызывается после метода `onStart()`, когда окно отображается на экране и пользователь может с ним взаимодействовать посредством графического интерфейса. Происходит инициализация переменной `mConnect`, которая устанавливает соединение с каким-либо устройством (рис. 3.3). Предположительно

это соединение было закрыто в методе onPause().

```
@Override  
public void onResume() { super.onResume();  
    if (mConnect == null) { connectDevice();  
    }  
}
```

Метод onPause() вызывается, если открывается новая Activity, а текущая помещается в стек неактивных. В этом методе необходимо высвободить ресурсы, занятые текущей активностью – выключить камеру, остановить воспроизведение потокового видео и т. д. Также важно в этом методе размещать быстрый легковесный код, в противном случае, если, например, при вызове данного метода осуществлять запись в БД каких-либо больших данных, процесс перехода к другой активности будет замедлен, что негативно скажется на отзывчивости программного интерфейса.

Метод onStop() вызывается, когда окно становится невидимым пользователю. Это произойдет в том случае, если была запущена другая активность. В этом методе можно размещать операции, которые требуют длительного времени на выполнение, например, обновление пользовательского интерфейса, чтение показаний датчика GPS. При нехватке памяти система Android может сама уничтожить скрытую Activity, минуя метод onStop() с вызовом метода onDestroy().

Метод onRestart() вызывается в тот момент, когда окно возвращается в приоритетный режим после вызова метода onStop(). Т. е. в момент, когда активность была остановлена и снова запущена. Метод onRestart() предшествует вызову метода onStart(). Данный метод можно использовать при повторном вызове активности для восстановления ее параметров и элементов пользовательского интерфейса.

Метод onDestroy() вызывается по окончании работы Activity посредством вызова метода finish() либо в момент уничтожения экземпляра активности системой Android.

Рассмотрим пример, представленный в прил. В. Необходимо создать новый

проект и добавить в него два файла: MainActivity.java и activity_main.xml. При запуске и остановке приложения нужно следить за консолью, в которую переопределенные методы будут писать сообщения (см. таблицу).

Порядок вызова методов в зависимости от действия пользователя

Действие пользователя	Порядок вызовов
Запуск приложения	onCreate() → onStart() → onResume()
Нажатие системной кнопки «Назад»	onPause() → onStop() → onDestroy()
Нажатие системной кнопки «Домой»	onPause() → onStop()
После нажатия кнопки «Домой» повторный вызов приложения	onRestart() → onStart() → onResume()
Экран телефона выключен	onPause() → onStop()
Экран телефона повторно включен	onRestart() → onStart() → onResume()

Понимание жизненного цикла Activity очень важно при разработке под ОС Android, так как это позволяет правильно строить логику приложения.

Вопросы:

1. Что такое элементActivity?
2. Перечислите элементы жизненного циклаActivity.
3. Какие методы Activity вызываются при смене ориентации устройства?
4. Каким образом происходит связывание интерфейса сActivity?

Лабораторная работа 4. Использование ресурсов приложения

В корне проекта существует директория ресурсных файлов res. Ресурсы хранятся в xml-файлах, в директории res/values.

При создании приложения в файл strings.xml добавляется первый ресурс app_name, значение которого есть имя приложения. В данном файле нужно

описывать строковые константы, которые будут использоваться в приложении (рис. 4.1). Требуется это для мультиязычности приложения и простоты доработки его в дальнейшем.

```
<resources>
  <string name="app_name">Hello world</string>
  <string name="sample">test string</string>
</resources>
```

Рис. 4.1. Пример файла strings.xml

При объявлении ресурса в xml файле имя должно быть уникальным, в классе R создается константа с таким же именем, чтобы мы могли иметь доступ к строковым элементам (рис.4.2).

```
public static final class string {
    public static final int abc_action_bar_home_description=0x7f060000;
    public static final int abc_action_bar_home_description_format=0x7f06000d;
    public static final int abc_action_bar_home_subtitle_description_format=0x7f06000e;
    public static final int abc_action_bar_up_description=0x7f060001;
    public static final int abc_action_menu_overflow_description=0x7f060002;
    public static final int abc_action_mode_done=0x7f060003;
    public static final int abc_activity_chooser_view_see_all=0x7f060004;
    public static final int abc_activitychooserview_choose_application=0x7f060005;
    public static final int abc_search_hint=0x7f06000f;
    public static final int abc_searchview_description_clear=0x7f060006;
    public static final int abc_searchview_description_query=0x7f060007;
    public static final int abc_searchview_description_search=0x7f060008;
    public static final int abc_searchview_description_submit=0x7f060009;
    public static final int abc_searchview_description_voice=0x7f06000a;
    public static final int abc_shareactionprovider_share_with=0x7f06000b;
    public static final int abc_shareactionprovider_share_with_application=0x7f06000c;
    public static final int abc_toolbar_collapse_description=0x7f060010;
    public static final int app_name=0x7f060011;
    public static final int sample=0x7f060012;
    public static final int status_bar_notification_info_overflow=0x7f060013;
}
```

Рис. 4.2. Класс R

Чтобы использовать значение ресурса, требуется в свойствах элемента ввести следующую конструкцию @string/имя_ресурса (рис. 4.3).


```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ru.omgtu.nabiullin.helloworld">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportRtl="true"
        android:theme="@style/AppTheme">

    </application>

</manifest>

```

Рис. 4.3. Пример файла AndroidManifest.xml

Кроме того, можно создать свой файл с константами, для этого следует нажать правой кнопкой на директорию values, выбрать New→Valuesresourcefiles и в созданном файле описать необходимое (рис.4.4).

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="tvTopText">Верхний текст</string>
    <string name="btnTopText">Верхняя кнопка</string>
    <string name="tvBottomText">Нижний текст</string>
    <string name="btnBottomText">Нижняя кнопка</string>
    <color name="llTopColor">#336699</color>
    <color name="llBottomColor">#339966</color>
</resources>

```

Рис. 4.4. Пример своего ресурсного файла

В нашем случае строковые ресурсы string и color находятся в одном файле, однако делать этого не стоит (здесь это сделано исключительно в учебных целях). Принято строковые константы и константы, которые задают цвет, описывать в разных файлах. Следом можно убедиться в том, что константы созданы. Для этого достаточно посмотреть файл R.java.

Если необходимо использовать значения ресурсов в коде программы, то их можно получить с помощью следующего метода:

```

getResources().getString(R.string.app_name);

```


Имена ресурсов, глобальных для всех файлов, хранятся в папке `res/values`. Т.е. невозможно в разных файлах создать ресурс с одним именем и типом.

Имена файлов ресурсов могут быть произвольными, и файлов можно создавать сколько угодно. В `R.java` попадут все ресурсы из этих файлов.

Вопросы:

1. Для чего необходимы файлы ресурсов приложения?
2. Как получить доступ к элементу файла ресурса приложения?
3. Можно ли вносить какие-либо изменения в файл `R.java`?

Лабораторная работа 5. Layout-файл в activity. Смена ориентации экрана

Для создания Activity необходимо выбрать пункт `File → New → Activity →` Необходимый тип Activity, в результате чего появится окно, в котором следует заполнить поле `ActivityName` (рис.5.1).

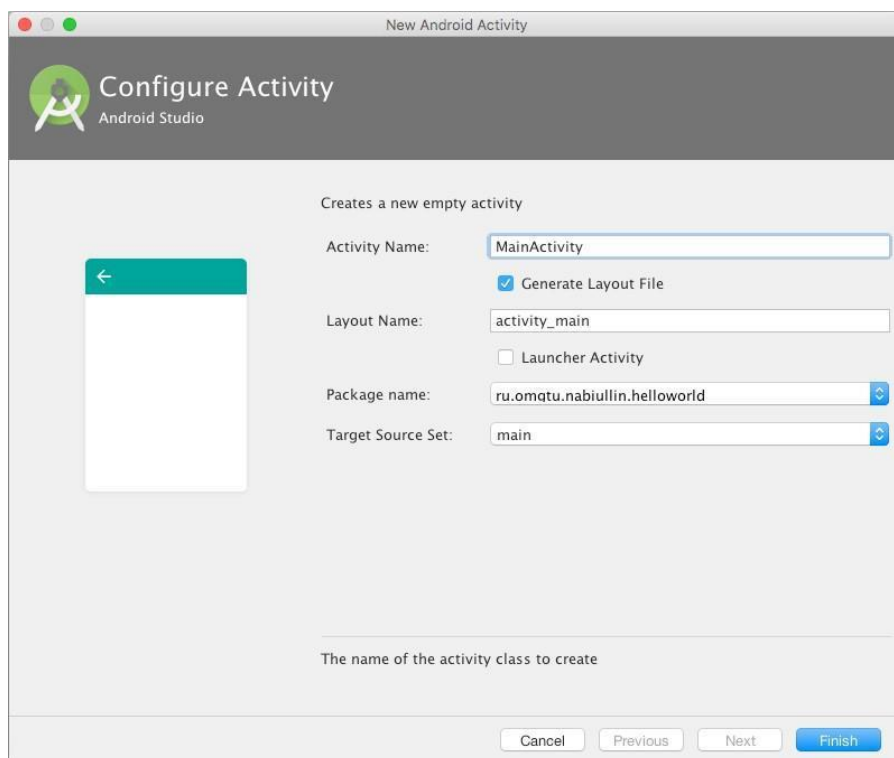


Рис. 5.1. Создание нового Activity

При создании Activity формируются layout-файл и класс с соответствующим

именем (рис. 5.2).

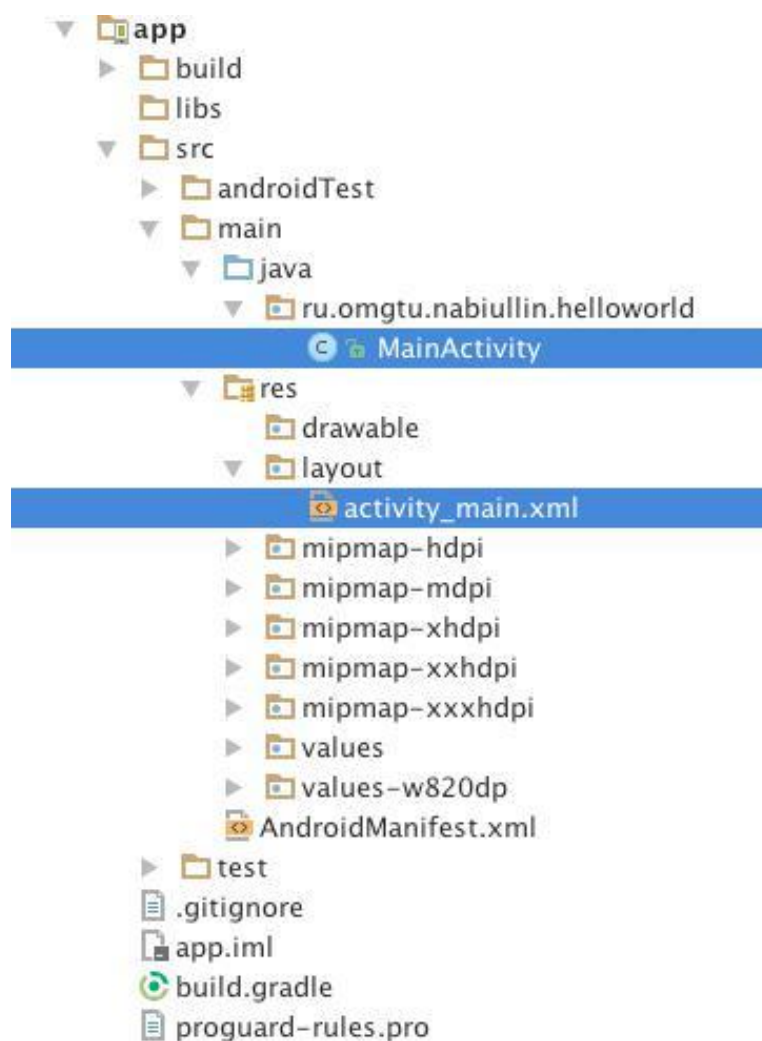


Рис. 5.2. Структура проекта

В новом классе MainActivity уже имеется метод onCreate (рис. 5.3). Данный метод вызывается методом setContentView, когда приложение создает и отображает Activity. Если запустить приложение, MainActivity отобразит то, что описано в activity_main.xml.

```

package ru.omgtu.nabiullin.helloworld;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}

```

Рис. 5.3. Класс MainActivity

Можно создать layout отдельно. Для этого необходимо выбрать пункт меню File → New → Android resource file, вписать название, выбрать тип ресурса и нажать ОК (рис. 5.4).

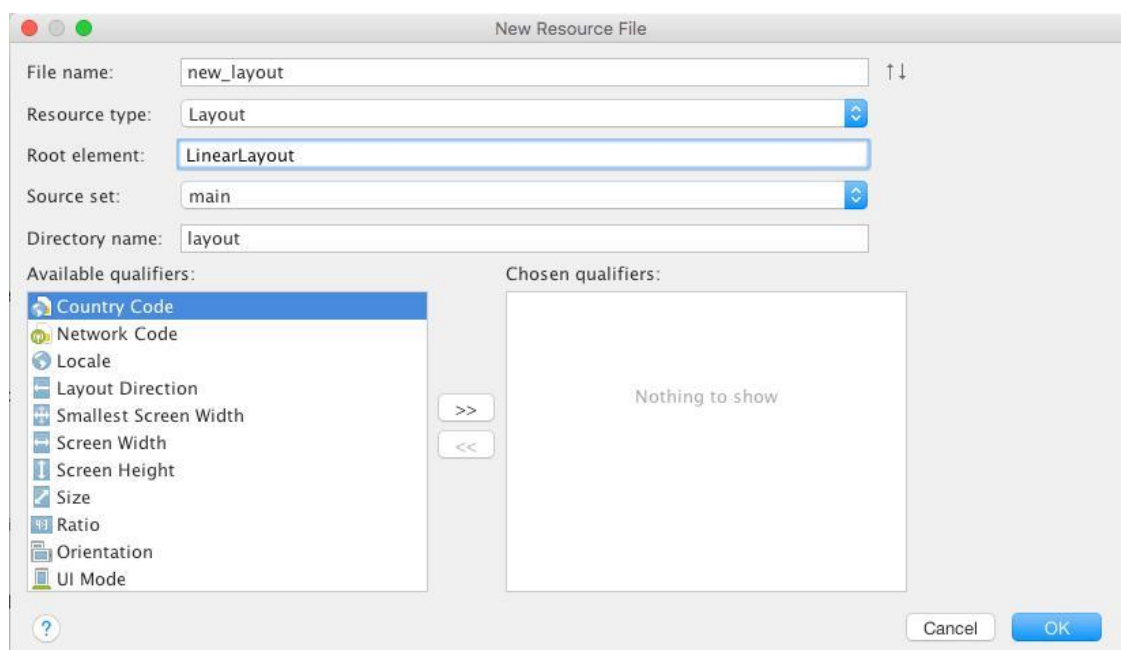


Рис. 5.4. Создание своего layout-файла

В результате этих манипуляций в папке с res/layout появится новый файл new_layout.xml. Добавим в него элемент TextView (рис. 5.5). После создания layout'а в классе R автоматически появилось упоминание данного layout'а.

```

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

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="New layout" />

</LinearLayout>

```

Рис. 5.5. Файл new_layout.xml

В методе onCreate изменим передаваемый параметр в метод setContentView на R.layout.new_layout:

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); setContentView(R.layout.new_layout);
}

```

При запуске приложения на экране будет отображен вновь созданный layout (рис. 5.6).

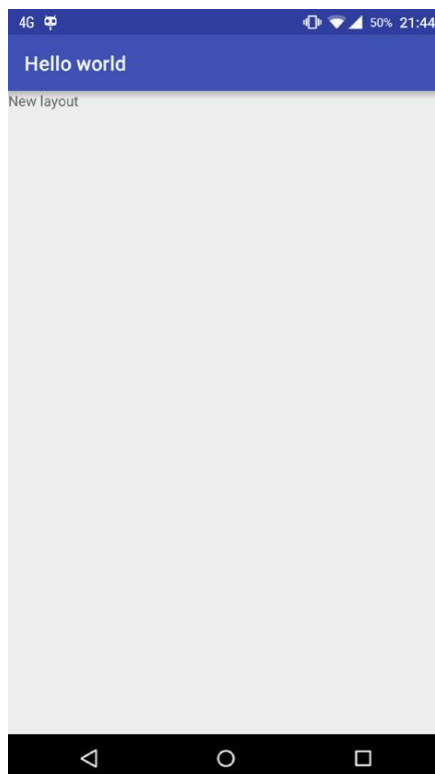


Рис. 5.6. Экран приложения

Layout при смене ориентации

Удалим TextView из нового layout'а и добавим в него вложенный вертикальный LinearLayout, а уже в нем разместим четыре кнопки (рис. 5.7).

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

    <LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <Button
            android:id="@+id/button1"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:text="Button1">
        </Button>
        <Button
            android:id="@+id/button2"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:text="Button2">
        </Button>
        <Button
            android:id="@+id/button3"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:text="Button3">
        </Button>
        <Button
            android:id="@+id/button4"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:text="Button4">
        </Button>
    </LinearLayout>
</LinearLayout>
```

Рис. 5.7. Файл new_layout.xml

Результат действий, осуществленных при запуске приложения, представлен на рис. 5.8.

В вертикальной ориентации все отображается нормально, но если перевести устройство в горизонтальную ориентацию, то не все элементы входят в рамки экрана (рис. 5.9).

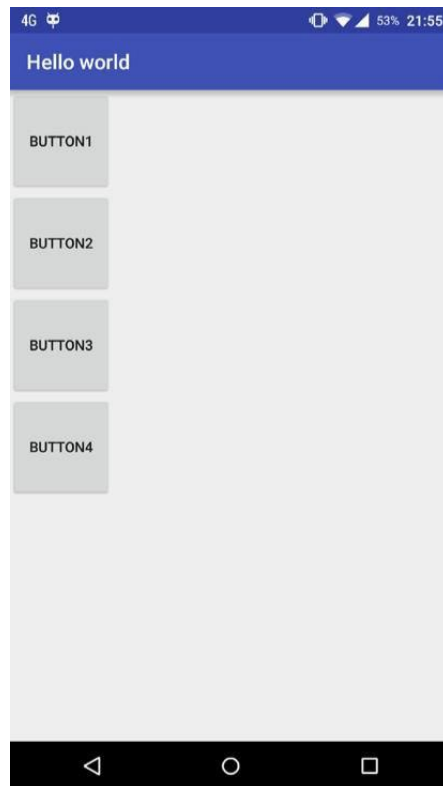


Рис. 5.8. Экран приложения в вертикальной ориентации

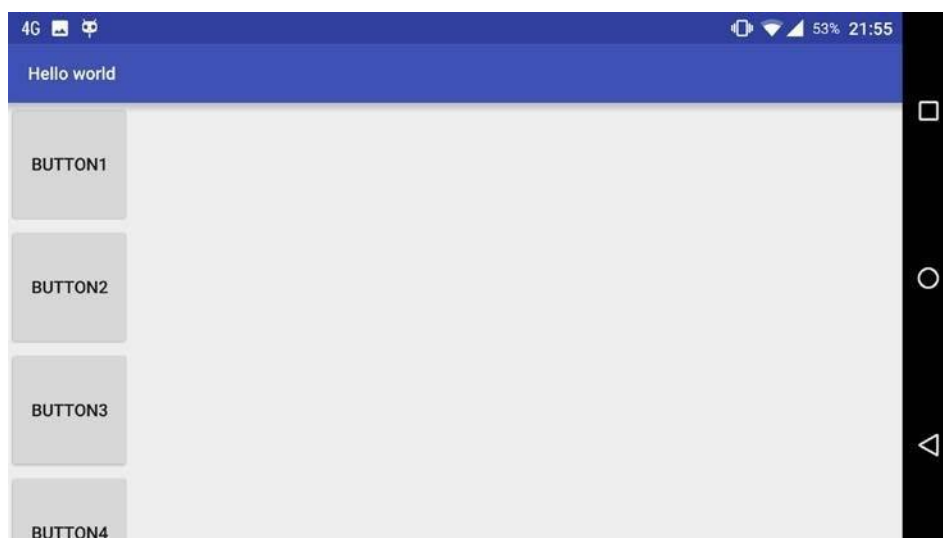


Рис. 5.9. Экран приложения в горизонтальной ориентации

Для решения этой проблемы необходимо создать еще один layout-файл с

идентичным именем, который будет подготовлен специально под горизонтальную ориентацию и в нашем случае выведет кнопки горизонтально. Для этого необходимо выбрать квалификатор Orientation (рис. 5.10).

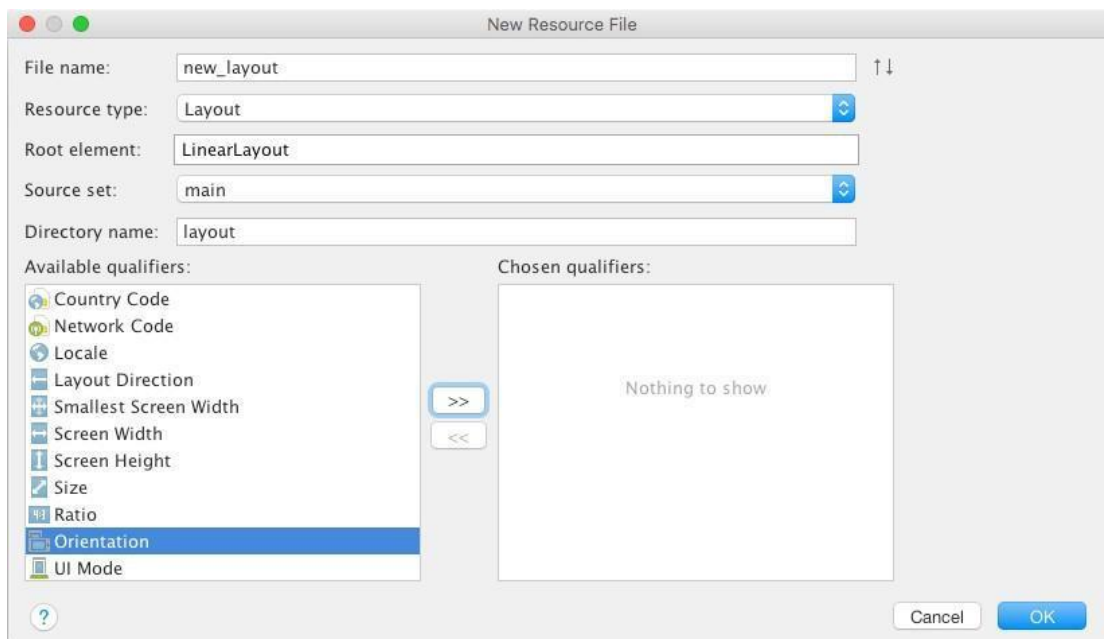


Рис. 5.10. Создание layout-файла с квантификатором

Выбрать Screenorientation = Landscape (рис. 5.11), так же изменится директория на layout-land.

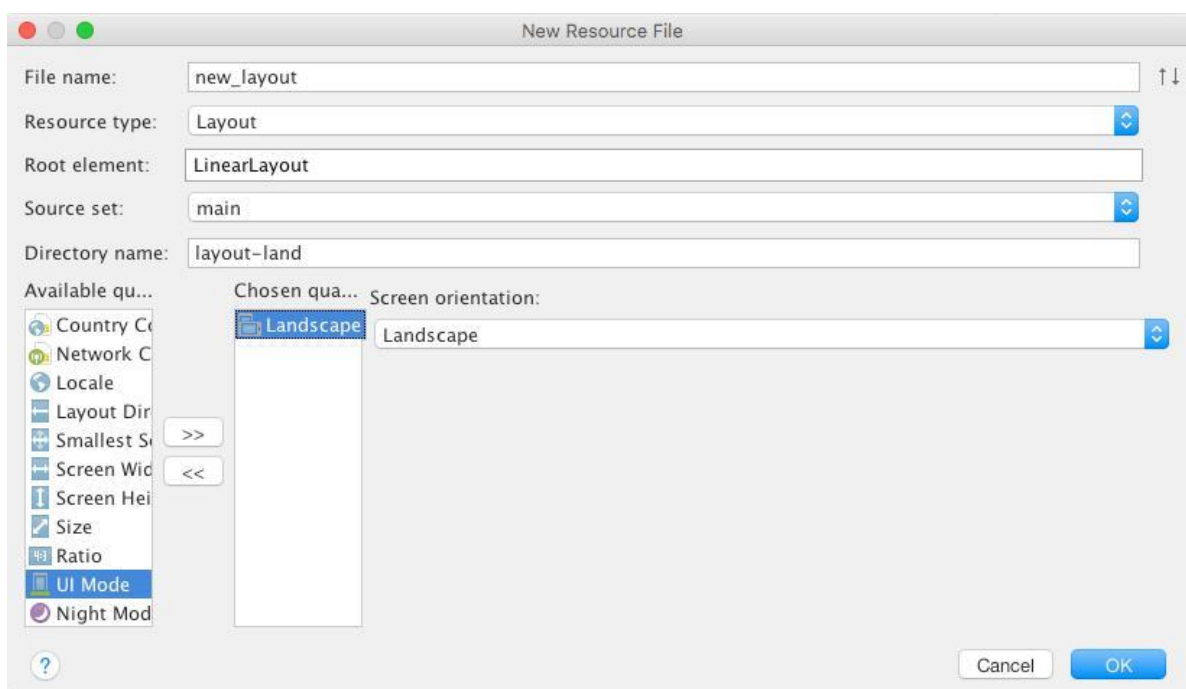


Рис. 5.11. Выбор ориентации экрана

Структура проекта с отображаемыми папками и ресурсами представлена под разные разрешения экрана (рис. 5.12).

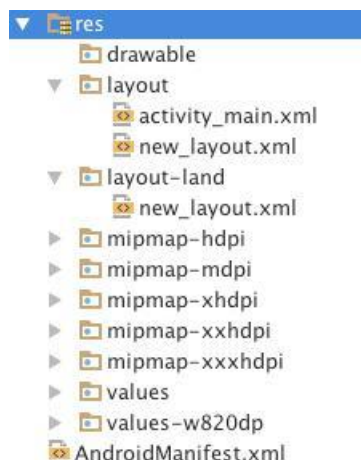


Рис. 5.12. Структура папки ресурсов

Далее необходимо добавить четыре кнопки во вложенном `LinearLayout`, только параметр ориентации следует указать как «горизонтальный» (рис.5.13).

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

    <LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <Button
            android:id="@+id/button1"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:text="Button1">
        </Button>
        <Button
            android:id="@+id/button2"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:text="Button2">
        </Button>
        <Button
            android:id="@+id/button3"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:text="Button3">
        </Button>
        <Button
            android:id="@+id/button4"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:text="Button4">
        </Button>
    </LinearLayout>
</LinearLayout>
```

Рис. 5.13. Файл `new_layout.xml` для горизонтальной ориентации

Activity читает layout-файл, который мы указывали в методе setContentView, т.е. new-layout.xml, и отображает его содержимое. При работе приложения учитывается ориентация устройства, и в случае горизонтальной ориентации используется файл new-layout.xml из папки res/layout-land. Запустив приложение, получаем результат (рис. 5.14).

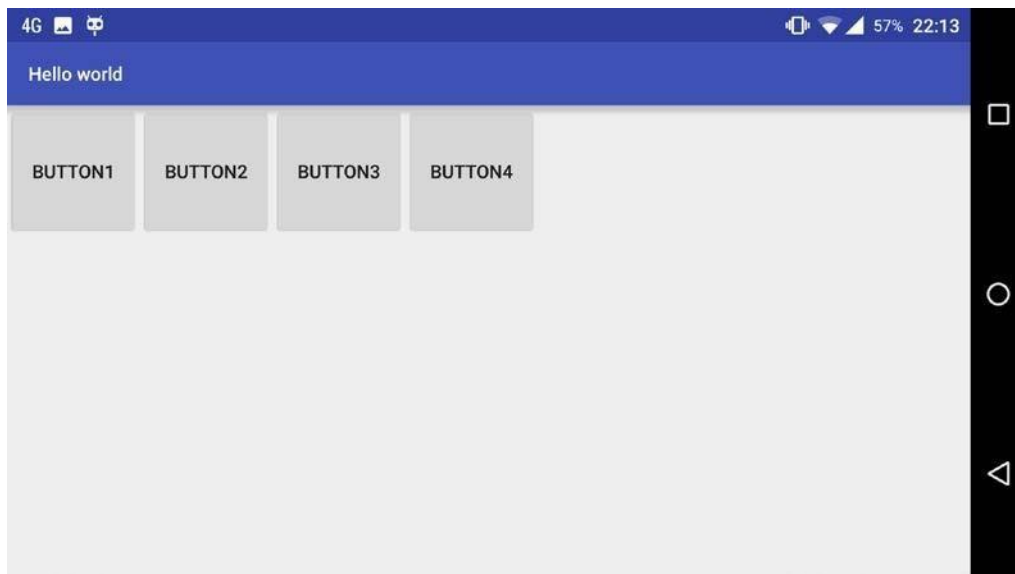


Рис. 5.14. Экран приложения в горизонтальной ориентации

Создавая layout-файл с альтернативной разметкой для горизонтальной ориентации экрана, необходимо использовать те же компоненты, которые используются в разметке для портретной ориентации. Если не добавить какой-либо элемент, то в портретной ориентации приложение будет работать, а в горизонтальной закроется с ошибкой.

Вопросы:

1. В каком каталоге приложения хранятся файлы ресурсов? Укажите полный путь.
2. Какое отличие имеет элемент <LinearLayout> от элемента <RelativeLayout>?

3. Какие действия в программе необходимо предусмотреть при смене ориентации экрана?

4. Можно ли иметь разные типы разметок в зависимости от разрешения устройства? Чем это обеспечивается?

Лабораторная работа 6. Всплывающие уведомления / toastnotification

ToastNotification – всплывающее уведомление, которое появляется поверх приложения и исчезает через некоторое время. Всплывающее уведомление также может быть создано службой, работающей в фоновом режиме. Обычно всплывающее уведомление содержит короткий текст уведомляющего характера.

Для создания уведомления необходимо инициализировать объект Toast при помощи метода `makeText()`. Для отображения на экране нужно вызвать метод `show()`.

```
Toast toast = Toast.makeText(getApplicationContext(),  
"Текст уведомления", Toast.LENGTH_SHORT);  
toast.show();
```

 У метода `makeText()` есть три параметра:

- контекст приложения;
- текстовое сообщение;
- продолжительность времени показа уведомления.

Константы продолжительности показа уведомления

Имеются две константы для указания продолжительности показа уведомления:

- `LENGTH_SHORT` – является константой по умолчанию, показывает уведомление в течение двух секунд;
- `LENGTH_LONG` – показывает уведомление в течение 3,5с.

Позиция на экране

По умолчанию место появления всплывающего уведомления – внизу. Для 62

изменения местоположения необходимо вызвать метод `setGravity()`, который принимает три параметра:

- константа для размещения объекта в пределах контейнера (`Gravity.CENTER`, `Gravity.TOP` и др.);
- смещение по оси X;
- смещение по оси Y.

Пример

Добавляем кнопку, присваиваем атрибуту `onClick` значение `showToast`.

Напишем метод `showToast`.

```
public void showToast(View view) {  
    Toast toast = Toast.makeText(getApplicationContext(), "Your  
    message", Toast.LENGTH_SHORT);  
    toast.setGravity(Gravity.CENTER, 0, 0);  
    toast.show(); }
```

Запустив приложение и нажав на кнопку, получим всплывающее уведомление по центру экрана, без смещений с указанным нами текстом, через две секунды после появления уведомление скрывается (рис. 6.1).

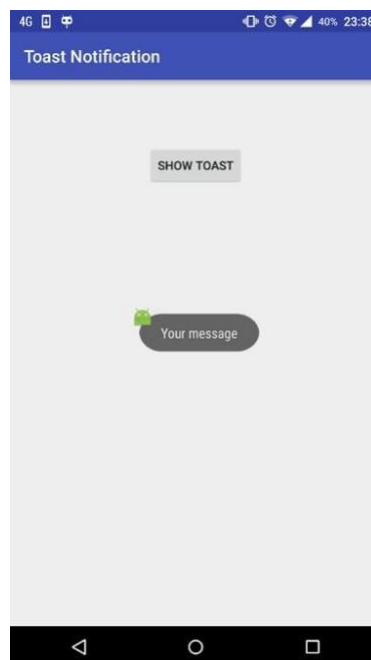


Рис. 6.1. Экран приложения с `ToastNotification`

Начиная с версии Android 4.4, всплывающие уведомления получили скругления, с версии Android 5.1 около уведомления отображается логотип приложения.

Если простого текстового сообщения недостаточно, можно создать собственный дизайн разметки уведомления. Для получения разметки из XML-файла и работы с ней в программе используется класс `LayoutInflater` и его методы – `getLayoutInflater()` или `getSystemService()`, которые возвращают объект `LayoutInflater`. Затем вызовом метода `inflate()` получают корневой объект `view` этой разметки. Например, для файла разметки уведомления с именем `notification_layout.xml` и его корневого элемента с идентификатором `android:id="@+id/toast_layout"` код будет таким:

```
LayoutInflater inflater = getLayoutInflater();
```

```
View layout = inflater.inflate(R.layout.notification_layout, (ViewGroup)
findViewById(R.id.toast_layout));
```

Код `notification_layout.xml`:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/toast_layout" android:orientation="vertical"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:padding="10dp" android:background="#DAAA">

    <ImageView android:id="@+id/imageView" android:layout_width="260dp"
    android:layout_height="190dp" android:layout_marginRight="10dp"
    android:background="@drawable/shrek_cat" />

    <TextView android:id="@+id/textView"
    android:layout_width="wrap_content" android:layout_height="match_parent"
    android:text="Your message" android:textColor="#FFF777" />

</LinearLayout>
```

И следом укажем методом `setView()` у объекта `toast`, укажем `Viewlayout` в качестве того контейнера, который мы хотим отобразить в всплывающем уведомлении, и покажем его (рис.6.2):

```
toast.setView(layout); toast.show();
```

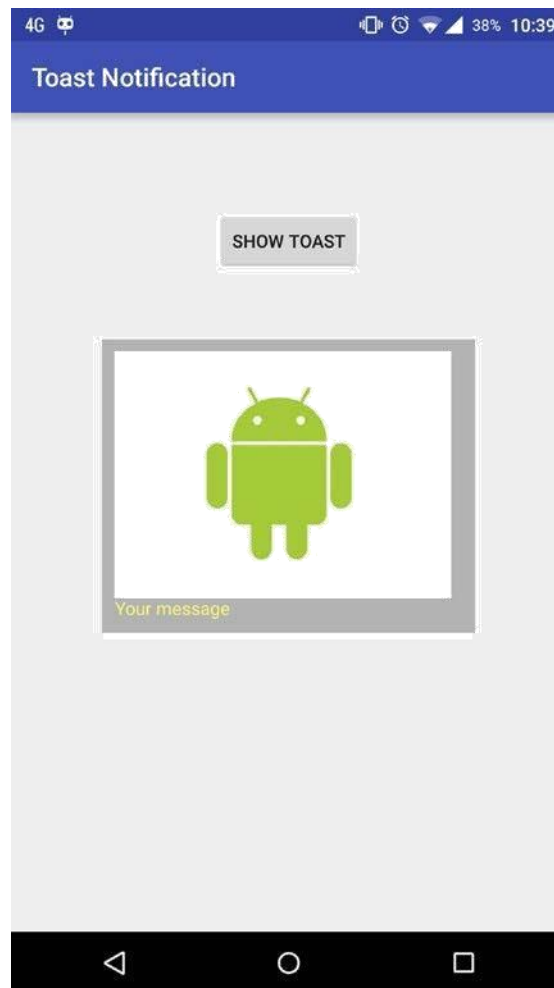


Рис. 6.2. Экран приложения с ToastNotification со своим

Layout Использование уведомлений Toast в рабочих потоках

Как элемент графического интерфейса Toast должен быть вызван в потоке GUI, иначе существует риск выброса межпоточкового исключения.

```
private void mainProcessing() {
    Thread thread = new Thread(null, doBackgroundThreadProcessing, "Background");
    thread.start();
}

private Runnable doBackgroundThreadProcessing = new Runnable() { public void run() {
    backgroundThreadProcessing();
}

};

private void backgroundThreadProcessing() { handler.post(doUpdateGUI);
}
```

```
// Объект Runnable, который вызывает метод из потока GUI private
Runnable doUpdateGUI = new Runnable() { public void run() {
    Context context = getApplicationContext(); String msg = "To open mobile development!"; int
duration = Toast.LENGTH_SHORT; Toast.makeText(context, msg,duration).show();
}
};
```

В листинге объект Handler используется для гарантии того, что уведомление Toast было вызвано в потоке GUI.

Вопросы:

1. Перечислите варианты при разработке приложения под ОС Android, в которых использование уведомлений Toast оправдано.
2. Можно ли сделать отображение Toast уведомления на базе собственной разметки? Каким классом это обеспечивается?
3. Каким параметром регулируется время отображения уведомления на экране?

Лабораторная работа 7. Уведомления / pushnotification

Уведомление – это сообщение, которое выводится поверх любого открытого приложения. Изначально часть уведомления появляется в отдельном контейнере сверху экрана либо в виде ярлыка в области уведомлений. Чтобы посмотреть полную информацию об обновлении, пользователю необходимо открыть панель уведомлений. Данные уведомления имеют больший приоритет важности в сравнении с всплывающими уведомлениями, поэтому будут зафиксированы в панели уведомлений до тех пор, пока пользователь не предпримет решения, что делать с уведомлением.

Стилизация уведомления описывается в объекте NotificationCompat.Builder. Чтобы создать уведомление, необходимо вызвать NotificationCompat.Builder.build(), который вернет объект Notification, содержащий описанные нами спецификации. Для отображения уведомления объект Notification передается в систему путем вызова метода NotificationManager.notify(), куда первым параметром необходимо

передать уникальный ID, вторым параметром – созданное уведомление.

Объект Notification должен содержать три элемента:

- небольшая иконка, задается с помощью метода `setSmallIcon()`;
- заголовок уведомления, задается с помощью метода `setContentTitle()`;
- текст уведомления, задается с помощью метода `setContentText()`.

Создадим простое уведомление, которое имеет в качестве иконки иконку приложения, особый текст и заголовок (рис. 7.1).

```
NotificationCompat.Builder mBuilder = (NotificationCompat.Builder)
new NotificationCompat.Builder(this)
.setSmallIcon(R.mipmap.ic_launcher)
.setContentTitle("Title notification")
.setContentText("Text notification");
NotificationManager mNotificationManager
=(NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mNotificationManager.notify(NOTIFY_ID, mBuilder.build());
```

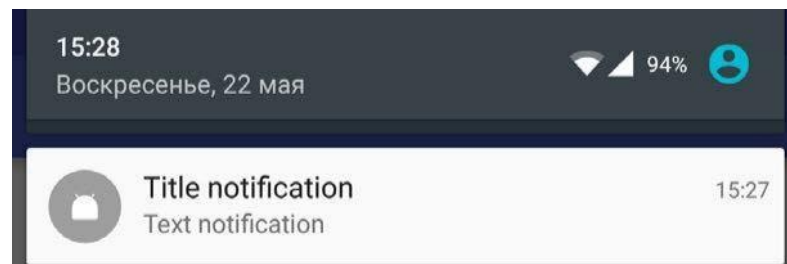


Рис. 7.1. Отображение простого Push-уведомления на панели уведомлений

Действия уведомлений

К уведомлениям можно привязать действия, которые позволяют пользователям упростить взаимодействие между уведомлением и приложением. Например, если нажать на уведомление приложения Календарь, откроется событие в приложении Календарь, уведомление о котором появилось в панели уведомлений. Начиная с версии Android 4.1, появилась возможность добавления в уведомления дополнительных кнопок.

Следует всегда определять действие, которое вызывается, когда пользователь нажимает на уведомление. Обычно это действие открывает Activity самого

приложения. Внутри класса Notification действие определяется объектом PendingIntent, содержащим объект Intent, который запускает операцию Activity из вашего приложения. Чтобы связать объект PendingIntent с жестом, вызовите соответствующий метод NotificationCompat.Builder. Например, если вам требуется запустить Activity, когда пользователь нажимает на текст уведомления в панели уведомлений, вы добавляете объект PendingIntent путем вызова метода setContentIntent().

```
NotificationCompat.Builder mBuilder = (NotificationCompat.Builder)
new NotificationCompat.Builder(this)
.setSmallIcon(R.mipmap.ic_launcher)
.setContentTitle("Title notification")
.setContentText("Text notification");

Intent resultIntent = new Intent(this, MainActivity.class); TaskStackBuilder stackBuilder =
TaskStackBuilder.create(this);                                stackBuilder.addParentStack(MainActivity.class);
stackBuilder.addNextIntent(resultIntent);

PendingIntent resultPendingIntent = stackBuilder.getPendingIntent(0,
PendingIntent.FLAG_UPDATE_CURRENT); mBuilder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
(NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mNotificationManager.notify(NOTIFY_ID, mBuilder.build());
```

Использование вспомогательных настроек

Перед отображением уведомления, для акцентирования внимания пользователя, можно использовать вибрацию, звуковой сигнал или активировать световой индикатор. Для этого необходимо передать константу в метод setDefault объекта Notification. Перечень доступных констант:

- Notification.DEFAULT_LIGHTS –светодиод;
- Notification.DEFAULT_SOUND – звуковоеуведомление;
- Notification.DEFAULT_VIBRATE – вибрация;
- Notification.DEFAULT_ALL – все возможныеварианты.

Уведомления с кнопками

В уведомлениях можно разместить до трёх кнопок. Кнопки помогут выбрать дальнейший сценарий приложения, например, загрузить другую Activity. Для добавления кнопки необходимо воспользоваться методом `addAction` объекта `Notification`. Добавим две кнопки к уведомлению и создадим для них действия. Первая кнопка будет открывать главную Activity, вторая – другую Activity (рис.7.2.).

```
Intent checkIntent = new Intent(this, MainActivity.class); PendingIntent checkPendingIntent =
PendingIntent.getActivity(this, 0, checkIntent,0);

Intent closeIntent = new Intent(this, SecondActivity.class); PendingIntent closePendingIntent =
PendingIntent.getActivity(this, 0, checkIntent,0);

NotificationCompat.Builder mBuilder = (NotificationCompat.Builder)
new NotificationCompat.Builder(this)
.setSmallIcon(R.mipmap.ic_launcher).setContentIntent(checkPendingIntent)
.setContentTitle("Titlenotification")
.setContentText("Textnotification")
.addAction(R.drawable.check, "Ok", checkPendingIntent)
.addAction(R.drawable.close, "Close", closePendingIntent); NotificationManager
mNotificationManager =
(NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mNotificationManager.notify(NOTIFY_ID, mBuilder.build());
```

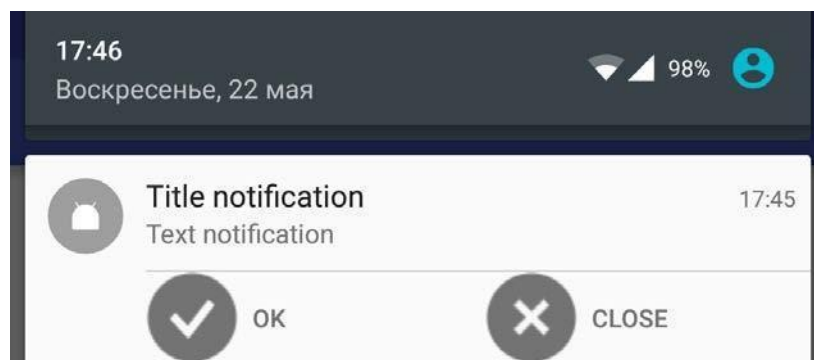


Рис. 7.2. Отображение Push-уведомления с кнопками на панели уведомлений

Приоритеты

Уведомления имеют такой параметр, как приоритет, суть его проста. Чем выше приоритет, тем выше в панели уведомлений будет находиться наше уведомление. Приоритет устанавливается методом `setPriority` у объекта `Notification` и принимает в качестве аргумента следующие константы:

- `Notification.PRIORITY_MIN`;
- `Notification.PRIORITY_LOW`;
- `Notification.PRIORITY_DEFAULT`;
- `Notification.PRIORITY_HIGH`;
- `Notification.PRIORITY_MAX`.

В Android 5.0 уведомления могут отображаться в плавающем окне, когда устройство разблокировано. Для того чтобы сделать ваше уведомление heads-up, необходимо у объекта `Notification` выставить высокий приоритет и добавить вибрацию.

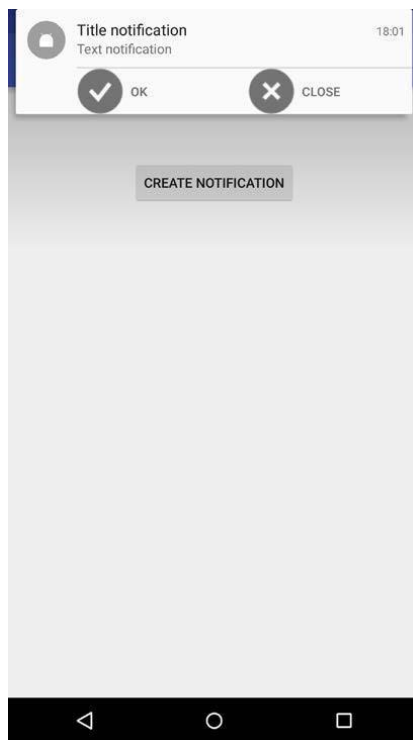


Рис. 7.3. Отображение Push-уведомления при заблокированном экране

Push-уведомления удобны, если приложение работает в фоновом режиме. Оно должно уведомлять пользователя о каких-либо важных событиях. Фоновое приложение просто создает уведомление, но не запускает Activity самостоятельно, это необходимо сделать пользователю в удобное для него время.

Вопросы:

1. Объясните принцип работы push-нотификаций.
2. Как с помощью программы возможно отследить была ли push-нотификация доставлена пользователю?
3. Как изменить разметку отображения push-нотификации?
4. Push-нотификация всегда должна отображаться на экране пользователя или возможно использование невидимых push-нотификаций?
5. Перечислите, что нужно иметь разработчику, чтобы начать использование push-нотификаций.

Лабораторная работа 8. Локализация приложения

Локализация дает возможность пользователям из разных стран запускать приложение на языке, который объявлен у них основным в настройках. Таким образом, если приложение запускается на телефоне с английской локализацией, то в интерфейсе приложения будут отображаться английские надписи, если с китайской локализацией, то иероглифы. Фактически это добавление других языков интерфейса в приложение.

Важно! Для того чтобы произвести локализацию, необходимо чтобы все тексты приложения, включая заголовки окон, надписи на кнопках, название самого приложения, хранились и использовались только с помощью строковых ресурсов.

Строковые ресурсы в Android приложении хранятся в папке `res/values/strings.xml` и выглядят следующим образом:

```
<string name="app_name">Local</string>
```

Прежде чем приступить к локализации, создадим несколько TextView для 71

отображения текста на экране приложения, создаем их в Layout'е activity_main нашего приложения:

```
<TextView
    android:textSize="20dp" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:text="@string/hworld" android:id="@+id/textView"
android:layout_alignParentTop="true" android:layout_alignParentStart="true" />
<TextView
    android:textSize="20dp" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:text="@string/text1" android:id="@+id/textView2"
android:layout_below="@+id/textView"
    android:layout_alignParentStart="true" />
<TextView
    android:textSize="20dp" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:text="@string/text2" android:id="@+id/textView3"
android:layout_below="@+id/textView2" android:layout_alignParentStart="true" />
```

После этого текст отобразится внутри TextView, используя строковые ресурсы, расположенные в файле values/strings.xml

```
<resources>
<string name="app_name">Local</string>
<string name="hworld">Hello World</string>
<string name="text1">Good evening, Android</string>
<string name="text2">Why I am not using iOS?</string>
<string name="text3">Learn as hard as you can</string>
</resources>
```

Существует несколько вариантов локализации приложения.

1. Файл ресурсов может быть задан вручную.

Для реализации такого подхода необходимо перейти в отображение папок для всего проекта. Затем открыть папку res, которая находится по адресу app/src/main/res. После этого создать Android resource directory (ПКМ → New → Android resource directory). Затем из списка Available qualifiers выбрать пункт Locate, а затем – интересующий нас язык.

После этих манипуляций появится папка values-ru, в случае если был выбран русский язык, если же был выбран английский, то префикс будет другим values-us.

Далее необходимо скопировать файл string.xml из папки values в папку values-ru и, уже непосредственно открыв его, задать перевод для каждой из строк.

Values-ru/string.xml

```
<resources>
<string name="app_name">Локализованное приложение</string>
<string name="hworld">Привет МИР!</string>
<string name="text1">Добрый вечер Андроид!</string> <string
name="text2">Почему я не пользуюсь iOS</string>
<stringname="text3">Учисьтакусердно,
кактолькоможешь</string> </resources>
```

2. Файл ресурсов может быть создан с помощью специального редактора TranslationEditor. Этот способ является более простым.

Необходимо выбрать файл string.xml из папки values и нажать на кнопку Openeditor. Для редактирования файла будет использоваться TranslationEditor.

Сам редактор очень прост в обращении, он представляет собой таблицу с ключом или названием строки (key), стандартными значениями (defaultvalue), а также поля для перевода строк на язык, который мы выбрали ранее. В редакторе можно в два клика добавить новую локализацию, например, мы хотим добавить поддержку французского языка. Жмем кнопку AddLocale и выбираем французский язык из списка (рис. 8.1).

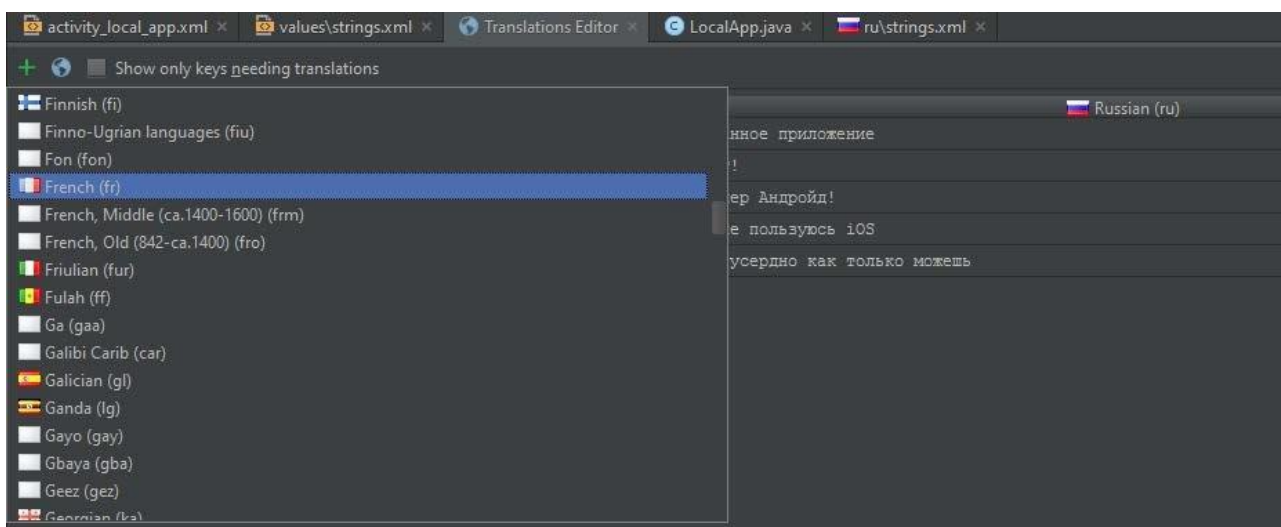


Рис. 8.1. Добавление языка в редактор TranslationEditor

Красным подсвечиваются те поля, для которых еще не было выбрано перевода (рис. 8.2).

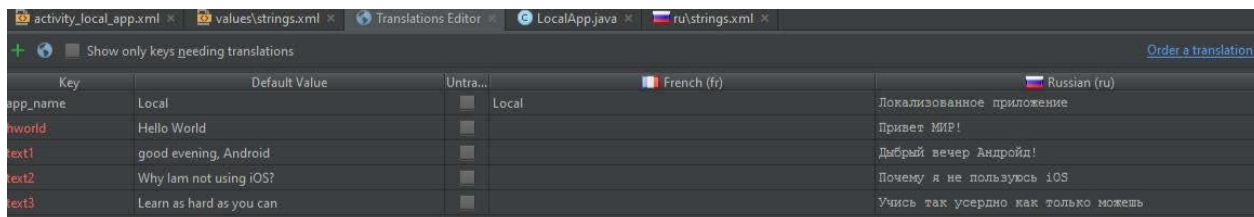


Рис. 8.2. Редактор TranslationEditor

Таким образом, используя первый или второй способ, можно локализовать приложение под любой язык. Язык в этом случае будет определяться автоматически по типу локализации устройства.

Вопросы:

1. Для каких целей используется локализация вприложении?
2. Каким образом целесообразно выбирать структуру хранения констант вфайлах?
3. Каким образом можно добавить локализацию к уже созданному приложению?

Лабораторная работа 9. Переключение между экранами

Приложение на Android не всегда состоит из одного экрана. Наверняка многие замечали, что если пользоваться Android приложением, то в нем обычно не один экран, а несколько, все это сделано, чтобы разбить на некоторые части приложение. Например, разграничить настройки и информацию об авторе от главного окна приложения.

Рассмотрим простое приложение под ОС Android, которое бы могло переходить по нажатию кнопки из главного окна приложения на другой экран. И

вследствие этого разберемся с основами переключения между экранами одного приложения.

Создадим новый проект в AndroidStudio и назовем его JumpWindow. Отметим Phone and Tablet и при выборе Activity укажем Empty Activity. Все остальные параметры оставим без изменений.

Добавим в макет нашего главного Activity(activity_main.xml) кнопку, при нажатии которой будет вызываться второй экран.

```
<Button
```

```
    android:layout_width="wrap_content"    android:layout_height="wrap_content"
    android:text="NextWindow" android:id="@+id/btnNextWindow" />
```

Теперь в классе MainActivity определяем данную кнопку:

```
Button btnNextWindow;
```

Присваиваем ей Activity в качестве обработчика:

```
Public class MainActivity extends AppCompatActivity implements View.OnClickListener
```

И в методе onCreate дописываем следующее:

```
    btnNextWindow = (Button) findViewById(R.id.btnNextWindow);
    btnNextWindow.setOnClickListener(this);
```

И теперь реализуем в методе onClick нажатие этой кнопки с помощью конструкции switch-case:

```
switch (v.getId()) {
    case R.id.btnNextWindow: break;
    default:
    break;
}
```

С помощью этого кода будем определять, когда была нажата кнопка, и переходить на другой экран.

Теперь создадим второй экран нашего приложения, для этого необходимо создать класс и прописать его в файле AndroidManifest.xml.

Чтобы создать новый Activity (новый экран), надо кликнуть кнопкой мыши по названию пакета, где расположен MainActivity, и выбрать New->Activity ->EmptyActivity (рис.9.1).

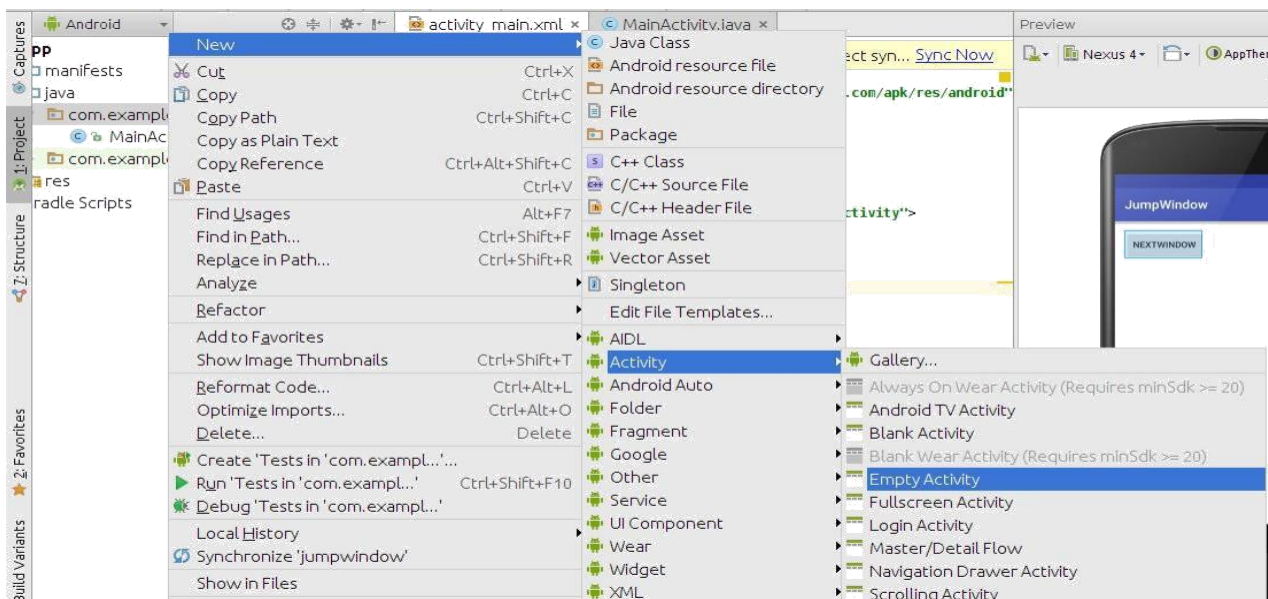


Рис. 9.1. Создание новой Activity

В новом окне необходимо ввести название новый Activity (рис. 9.2).

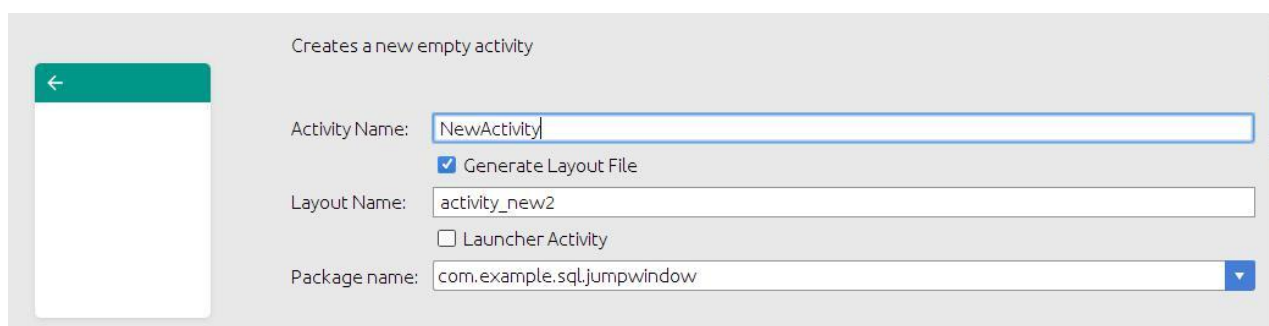


Рис. 9.2. Описание новый Activity

Новый Activity создан. Причем AndroidStudio сам создает java класс Activity и xml файл макета новой Activity.

Необходимо открыть файл AndroidManifest.xml, в него автоматически должен добавиться вновь созданный класс Activity.

Теперь перейдем в файл макета нового Activity и добавим в него элемент TextView:


```
<TextView
```

```
    android:layout_width="match_parent"    android:layout_height="wrap_content"  
    android:text="@string/txtAct2" android:textSize="50sp" />
```

Открыв MainActivity, добавим в конструкцию switch-case, которая находится в методе onClick и с помощью которой осуществляется обработка нажатия кнопки btnNextWindow, и функцию вызова нового экрана.

Создаем объект Intent, у которого в качестве входных параметров используются контекст и класс второй Activity, затем методом startActivity запускаем наш Intent.

```
Intent intent = new Intent(this, NewActivity.class); startActivity(intent);
```

Приложение готово (рис. 9.3). Исходный код можно найти в прил. Г.

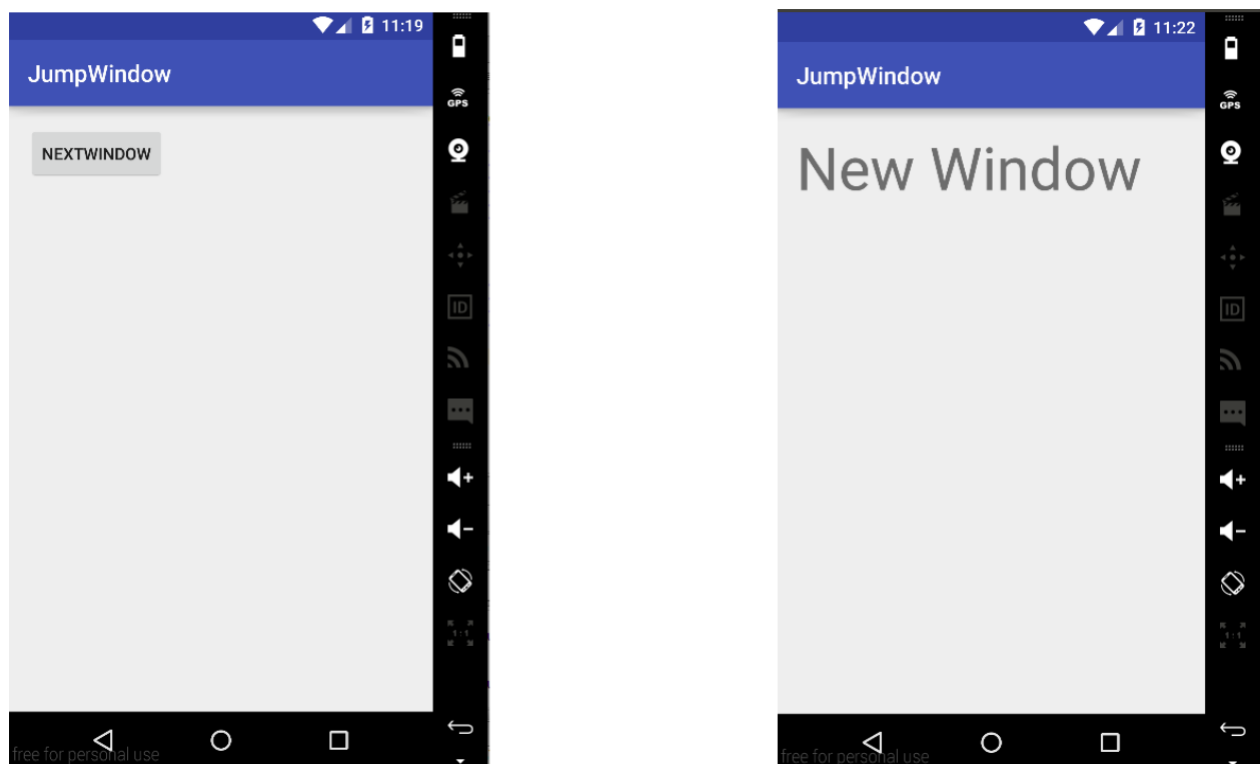


Рис. 9.3. Экраны приложения

Передача данных между экранами

Этот пример является простейшим примером вызова другого экрана. Иногда требуется не только вызвать другой экран активности, но и передать данные, которые понадобятся в дальнейшем для его работы. Например, имя пользователя, e-mail и т. д. В этом случае нужно задействовать область `extraData` в классе `Intent`. Эта область представляет собой список параметров ключ-значение, который передается на новый экран. В качестве ключей в этой области используются строки, а в качестве значений – примитивные данные.

Для передачи параметров используется метод `putExtra()`. Новая `Activity` должна вызвать подходящий метод: либо `getStringExtra()`, либо любой подобный, используя созданный `Intent`, в качестве объекта для которого вызывается метод. Выбор метода зависит от того, какой вид данные должны будут приняты в новой `Activity` (строковый, целочисленный и т. д.).

Вопросы:

1. Каким образом осуществляется переход между «окнами» в приложении на ОС Android?
2. За что отвечает класс `Intent` в Android-приложении?
3. Какие типы данных можно передавать через область `extraData` в классе `Intent`?

Лабораторная работа 10. Организация сервиса в приложении

Сервис (`service`) переводится как служба. Под сервисом будем понимать некую задачу, которая должна отработать в фоновом режиме и для которой не нужны элементы пользовательского интерфейса. Сервис может быть запущен и остановлен как из приложения (`Activity`), так и из других сервисов. В качестве примера сервиса можно рассмотреть почтовую программу, где сервис будет проверять наличие почты и в случае получения новой почты, выдавать

нотификацию пользователю. При этом такой сервис можно запустить, и он будет работать независимо от приложения (Activity). Другими словами, приложение, которое запустило сервис, может быть уже закрыто, а сервис будет доступен и будет обрабатывать получение сообщений.

Для примера работы с сервисом нужен минимально необходимый пользовательский интерфейс (рис. 10.1), код для запуска сервиса (рис. 10.2) и код самого сервиса (рис.10.3).

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:orientation="vertical">
    <Button
        android:id="@+id/startButton" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:onClick="onClickStart" android:text="@string/start
        service">
    </Button>
    <Button
        android:id="@+id/stopButton" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:onClick="onClickStop" android:text="@string/stop
        service">
    </Button>
    <ProgressBar
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:indeterminate="true">
    </ProgressBar>
</LinearLayout>
```

Пример файла activity_main.xml

```
package example;

import android.app.Activity; import android.content.Intent; import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity { final String LOG_TAG = "MainActivity";
    public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}
```

```

public void onClickStart(View v) {
startService(new Intent(this, MyService.class));
}
public void onClickStop(View v) {
stopService(new Intent(this, MyService.class));
}
}

```

Исходный код активности, запускающей сервис

```

packageexample;

import android.app.Service; import android.content.Intent; import android.os.IBinder;
import android.util.Log;

public class MyService extends Service { final String LOG_TAG = "MyService"; public
void onCreate() {
    super.onCreate(); Log.d(LOG_TAG, "onCreate");
}
    public int onStartCommand(Intent intent, int flags, int startId) { Log.d(LOG_TAG,
"onStartCommand");
        someTask();
        return Service.START_STICKY;
    }
    public void onDestroy() { super.onDestroy(); Log.d(LOG_TAG, "onDestroy");
    }
    public IBinder onBind(Intent intent) { Log.d(LOG_TAG, "onBind"); return null;
    }
    voidsomeTask() {
    }
}

```

Исходный код сервиса

Рассмотрим методы, реализованные в сервисе. Методы onCreate(), OnDestroy() работают по тому же принципу, что и подобные методы в активности. Метод onBind(Intentintent) в нашем случае не используется, при этом по умолчанию он возвращает значение null. Метод onStartCommand(Intentintent, intflags, intstartId) принимает в качестве аргументов три параметра – это экземпляр интента, флага и

id-процесса. Поскольку сервисы запускаются в главном потоке приложения, любые операции, выполняющиеся в методе `onStartCommand(Intent intent, int flags, int startId)`, будут осуществляться в контексте главного потока. Поэтому если требуется выполнить какие-либо операции в фоновом потоке, его сервис следует запустить из этого метода. Константа, возвращаемая методом `onStartCommand()`, может иметь одно из следующих имен:

START_STICKY– если будет возвращена эта константа, метод `onStartCommand()` будет вызываться при повторном запуске сервиса после преждевременного завершения работы. Подобный режим используется для сервисов, которые сами обрабатывают свои состояния. Такие сервисы завершают свою работу при необходимости. Например, к таким сервисам можно отнести сервисы, проигрывающие музыку.

START_NOT_STICKY– в режиме, описанном этой константой, сервисы запускаются для выполнения отдельных команд. При этом сервисы останавливают свою работу с использованием метода `stopSelf()`, как только команда выполнена.

START_REDELIVER_INTENT– константа, описывающая комбинацию предыдущих двух режимов. Сервис будет запущен повторно только в том случае, если система преждевременно завершила работу. В случае если сервис завершился до вызова метода `stopSelf()`, вызовется обработчик `onStartCommand()`, который продолжит обработку события, не завершенную должным образом с предыдущего раза.

По параметру `flags` можно узнать, какой тип процесса был запущен.

START_FLAG_REDELIVERY– константа, которая указывает, что параметр `Intent` был повторно передан при принудительном завершении работы сервиса перед явным вызовом метода `stopSelf()`.

START_FLAG_RETRY– константа, которая показывает, что сервис был запущен после аварийного завершения работы. Данный параметр передается, если сервис работал в режиме *START_STICKY*.

Значение параметра `flags` можно использовать для определения характера запуска: аварийный или в обычном режиме (рис. 10.4).

```

@Override
public int onStartCommand(Intent intent, int flags, int startId) { if ((flags
& START_FLAG_RETRY) == 0) {
    // Повторный запуск
}
else {
    // Выполнение действий
}
returnService.START_STICKY;
}

```

Пример метода проверки аварийного завершения работы сервиса

В случае повторного запуска после аварийного завершения можно выполнить определенные действия в программе. Если же аварийного случая завершения не было, можно осуществить запланированные действия в системе.

Вопросы:

1. Что такое сервис в Androidприложении?
2. Чем сервис отличается отActivity?
3. Опишите ситуацию, в которой применение сервиса будет целесообразным?
4. Каким способом можно организовать автоматически перезапускаемыйсервис?

Лабораторная работа 11. Сохранение данных в приложении

Существует несколько способов для сохранения данных:

- Sharedpreferences (настройки) – служит для хранения примитивных типов данных в виде парключ–значение;
- внутренняя память – служит для хранения данных во внутренней памятиустройства;
- внешняя память – служит для хранения данных во внешней памяти

устройства;

- база данных SQLite – служит для хранения структурированных данных в БД.

SharedPreferences

Способ реализуется при помощи стандартного класса SharedPreferences, разработанного специально для сохранения настроек. Файл создается приложением в своей папке автоматически, процесс работы с файлом полностью оптимизирован.

SharedPreferences поддерживает следующие типы значений:

- boolean;
- int;
- long;
- float;
- String.

Данные сохраняются парами с ключом типа String. Настройки можно загрузить в приложение двумя методами:

- `getSharedPreferences(Stringname, intmode)` – используется, когда требуется сохранять и использовать несколько файлов настроек. Файлы определяются именем и доступны из любой активности приложения;
- `getPreferences(intmode)` – используется, когда требуется только один файл настроек, доступный только внутри текущей активности.

Оба метода возвращают экземпляр класса SharedPreferences. Эти методы вызываются внутри метода `onCreate()`.

В качестве модификаторов доступа (`mode`) используются три стандартные константы:

- `Context.MODE_PRIVATE` – не предоставляет доступа к файлу извне;
- `Context.MODE_WORLD_READABLE` – позволяет получать доступ к файлу для чтения вне приложения;
- `Context.MODE_WORLD_WRITEABLE` – позволяет получать доступ к файлу для записи вне приложения.

Сохранять информацию обычно необходимо при закрытии приложения, лучше всего для этого подходят методы `onPause()` или `onStop()`.

Для внесения изменений в настройки необходимо получить объект типа `SharedPreferences.Editor`. Для этого у объекта типа `SharedPreferences` вызывается метод `edit()`. Для добавления значений используются методы типа:

- `putBoolean(String key, booleanvalue);`
- `putInt(String key, int value)` и др.

Для удаления всех записей используют метод `clear()`, а методом `remove(Stringkey)` отмечается запись дляудаления.

Для сохранения настроек используются методы:

- `apply()` – ничего невозвращает;
- `commit()` – возвращает подтверждение об успешнойзаписи.

Записанные значения требуется прочитать при повторном запуске приложения, обычно в методе `onResume()`. Для того чтобы проверить наличие записанного объекта, используется метод `contains(Stringkey)`. Для чтения значений используются такие методы, как:

- `getBoolean(String key, booleandefValue);`
- `getInt(String key, int defValue)` и др.

Для того чтобы извлечь все записи из файла, используется метод `Map<String,?>getAll()`.

Используя метод `registerOnSharedPreferenceChangeListener()`, можно зарегистрировать callback, который будет реагировать на изменения настроек.

Пример приложения для работы с файлами `Sharedpreferences` приведен в прил. Д.

Внутренняя память

У каждого приложения существует собственная папка (`/data/data/apps_dir/files`), в которую можно сохранять свои файлы. Другие приложения по умолчанию не имеют доступа к этим файлам. Все файлы, созданные таким 84

способом, удалятся вместе с приложением.

Для открытия и создания файла используется стандартный метод `FileOutputStreamContext.openFileOutput(StringfileName, intmode)`. Запись в файл осуществляется методом `write()`.

Открыть файл для чтения можно с помощью метода `FileInputStreamContext.openFileInput(StringfileName)`. Чтение данных из файла осуществляется методом `read()`.

Для закрытия файлов применяют метод `close()`.

Для работы с внутренней памятью существует несколько специальных методов:

- `Context.getCacheDir()` – используется для сохранения кэша приложения;
- `Context.getFilesDir()` – позволяет получить абсолютный путь к файлам приложения;
- `Context.mkdir(StringdirName, intmode)` – служит для создания при необходимости новой директории во внутренней памяти приложения;
- `Context.deleteFile(StringfileName)` – позволяет удалить ненужный файл;
- `Context.listFiles()` – возвращает массив имен файлов, сохраненных во внутренней памяти приложения.

Пример работы с файлами из внутренней памяти приложения в прил. Д.

Внешняя память

Внешней памятью считается SD-карта или встроенная память. Файлы, сохраненные во внешнюю память, могут быть получены и изменены любым приложением. Для того чтобы получить разрешение работать с SD-картой, в манифесте требуется указать разрешение `android.permission.WRITE_EXTERNAL_STORAGE`.

Для проверки доступности внешней памяти используют метод `Environment.getExternalStorageState()`. Он возвращает различные состояния, но для нас интересны только:

- `Environment.MEDIA_MOUNTED` – означает, что память полностью готова к работе;

- `Environment.MEDIA_MOUNTED_READ_ONLY` – означает, что память позволяет только читать файлы.

При любом другом состоянии работа с памятью невозможна.

Чтобы получить путь к директории внешней памяти, используют метод `FileEnvironment.getExternalStorageDirectory()`. Кроме того, можно использовать метод `FileContext.getExternalFilesDir(String type)`. В качестве параметра `type` передается тип возвращаемой директории:

- `null`;
- `Environment.DIRECTORY_MUSIC`;
- `Environment.DIRECTORY_MOVIES` и др.

Для работы с файлами используются стандартные средства Java.

Вопросы:

1. В каких случаях целесообразно использовать сохранение данных во внутренней памяти, а в каких – во внешней?
2. С помощью каких методов можно создать объект `SharedPreferences`? В чем их различия?
3. Какие методы используются для сохранения файла во внутренней памяти?
4. Какие существуют типы директорий во внешней памяти?

Лабораторная работа 12. Знакомство с SQLite. Хранение данных при помощи SQLite

Особенности SQLite

Системой управления базой данных SQLite является библиотека, в процессе применения которой используется SQL движок базы данных. Код SQLite

находится в свободном доступе, что позволяет бесплатно использовать его в

коммерческих или частных целях. Отличительная особенность SQLite – она не имеет отдельного процесса сервера, это значит, что SQLite считывает и записывает информацию непосредственно на обычные дисковые файлы. Полная база данных SQL с несколькими таблицами, индексами, триггерами и представлениями, содержится в одном файле на диске. Формат файла базы данных является кроссплатформенным – значит, можно свободно копировать базу данных с 32-битных на 64-битные системы, и наоборот.

Создание приложения

Для более детального знакомства и понимания основных принципов построения БД и работы с ней напомним простое приложение – телефонный справочник, который будет хранить имя и номер телефона. По нажатию кнопки Record будет производиться запись в базу данных имени и номера, по нажатию кнопки Reading будет выводиться вся база контактов в лог приложения, по нажатию кнопки Removal будут удаляться все записи в базе.

Для написания приложения создадим новый проект:

1. Укажем Application name: SQLiteExample.
2. Выберем Phone and Tablet.
3. Выберем EmptyActivity, дальше все оставим без изменений.

Откроем файл макета (activity_main.xml). Удалим RelativeLayout и вместо него напомним LinearLayout. Поместим в него два текстовых поля (EditText) и три кнопки. Кнопки поместим в отдельный LinearLayout, чтобы они выровнялись по одной линии. Зададим полям и кнопкам имена (рис.12.1).



Рис. 12.1. Файл макета

Пример этого кода можно посмотреть в прил. Е.

Теперь в главном классе Activity пропишем элементы экрана, найдем кнопки, пропишем обработчики кнопок и найдем поля ввода.

В методе onClick при нажатии на любую кнопку считываем значение текстовых полей и сохраняем в переменные типа string. Далее напишем конструкцию switch для разделения действий по отдельным кнопкам.

Теперь создадим класс для работы с базой данных. Назовем его BDWork.

Он должен быть унаследован от абстрактного класса SQLiteOpenHelper. Для этого напишем **public class DBWork extends SQLiteOpenHelper**. Это нужно для того, чтобы реализовать работу с базой данных (открытие, добавление, удаление).

В этом классе должно быть реализовано два обязательных метода:

- 1) onCreate – вызывается при первом создании базы данных;
- 2) onUpgrade – вызывается при изменении базы данных.

Также в этом классе нужно реализовать конструктор. Вызываем конструктор SuperClass и передаем ему четыре параметра: контекст, имя базы данных, нулевой и версию базы данных.

```
public DBWork(Context context, String name, int version) {
```

```

super(context, name, null, version);
}
@Override
public void onCreate(SQLiteDatabase db) {
}
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
}

```

Дальше нам нужно прописать константы для версии базы данных, имени БД и имени таблицы, а также указать номер версии БД, начиная со значения 1, по изменению номера версии класс DBWork будет понимать, что структуру БД нужно обновить. Проверка номера версии реализуется в методе onUpgrade.

```

public static final int DATABASE_VERSION = 1;
public static final String DATABASE_NAME = "phoneDb"; public static final String
TABLE_CONTACTS = "contacts";

```

Также зададим константы для заголовков столбцов

```

public static final String KEY_ID = "_id"; public static final String KEY_NAME = "name";
public static final String KEY_NUMBER = "number";

```

После задания констант нужно поправить конструктор, где будут использоваться наши константы.

```

super(context, DATABASE_NAME, null, DATABASE_VERSION);

```

Теперь в методе onCreate создадим таблицы и заполним их начальными данными с помощью специальных команд SQL.

```

public void onCreate(SQLiteDatabase db) { db.execSQL("create table " +
TABLE_CONTACTS + "(" +
KEY_ID
+ " integer primary key," + KEY_NAME + " text,"
+ KEY_NUMBER + " text" + ")");
}

```

При написании команд и ключей, необходимо между ними вставлять пробелы, чтобы получить корректный SQL запрос.

В методе onUpgrade, который сработает при изменении номера версии можно

реализовать запрос в БД на уничтожение таблицы (**droptable**), после чего вновь вызвать метод onCreate для создания новой версии таблицы с обновленными структурой.

```
public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
db.execSQL("drop_table " + TABLE_CONTACTS); onCreate(db);
}
```

После создания БД в классе MainActivity будем сохранять данные в базу и считывать их оттуда.

Объявляем переменную класса dbWork (DBWork**dbWork**;;), создаем его экземпляр в методе onCreate(dbWork = **new**DBWork(**this**;;), в методе onClick создаем объект класса SQLiteDatabase(SQLiteDatabasedatabase = **dbWork.getWritableDatabase**();) этот класс переназначен для управления базой данных SQLite. Затем нужно импортировать этот класс (**import**android.database.sqlite.SQLiteDatabase;;).

В классе SQLiteDatabase определены следующие методы:

- query() – для чтения данных изБД;
- insert() – для добавления данных вБД;
- delete() – для удаления данных изБД;
- update() – для изменения данных вБД;
- execSQL() – для выполнения любого кода на языке SQL относительно базыданных.

Далее создадим объект класса ContentValues(ContentValuescontentValues = **new**ContentValues();) – этот класс используется для добавления новых строк в таблицу. И также запишем его в **import**android.content.ContentValues;.

Далее напишем в case в качестве обработчика кнопки add следующие строки:

```
contentValues.put(DBHelper.KEY_NAME, name);//заполнение content.Values
попарно //name, contentValues.put(DBHelper.KEY_MAIL, number);// номер
database.insert(DBWork.TABLE_CONTACTS, null, contentValues);//методом insert
//вставляем строки в таблицу
```

С помощью обработчика Read реализуем чтение всех записей в таблице – методом query (Cursor cursor = database.query(DBWork.**TABLE_CONTACTS**, null, null, null, null, null);). На вход методу подается имя таблицы, список запрашиваемых полей, сортировка и группировка. В нашем случае сортировка и группировка не требуются.

Метод query возвращает объект класса cursor – его можно рассматривать как набор строк с данными. Далее делаем проверку наличия записей в объекте класса cursor и проверяем порядковые номера столбцов cursor по их именам с помощью метода getColumnIndex. Позже эти номера используем для чтения данных в методах getInt и getString и выводим данные в log.

При помощи метода moveToNext перебираем все строки в cursor. Если записи в cursor отсутствуют, выводим в log соответствующее сообщение. Далее закрываем cursor методом close(), тем самым освобождая занимаемые им ресурсы.

caseR.id.btnReading:

```
Cursor cursor = database.query(DBWork.TABLE_CONTACTS,
null, null, null, null, null, null); if (cursor.moveToFirst()) {
intidIndex = cursor.getColumnIndex(DBWork.KEY_ID);
intnameIndex = cursor.getColumnIndex(DBWork.KEY_NAME);
intnumbIndex = cursor.getColumnIndex(DBWork.KEY_NUMBER);
do{
    Log.d("mLog", "ID = " + cursor.getInt(idIndex) + ", name = " +
cursor.getString(nameIndex)+ ", numb = " + cursor.getString(numbIndex));
    } while (cursor.moveToNext());
} else
    Log.d("mLog", "0 rows"); cursor.close();
break;
```

Затем создаем кнопку очистки таблицы методом delete. На вход передаем имя таблицы и null. Последний обозначает, что из таблицы нужно удалить все данные.

caseR.id.btnRemoval: database.delete(DBWork.**TABLE_CONTACTS**, null, null); **break**;

Внешний вид приложения отражен на рис. 12.3.

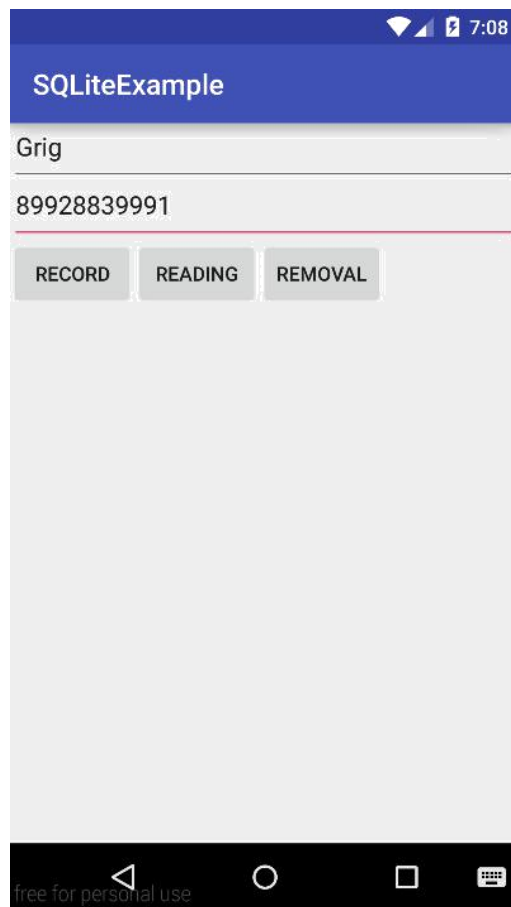


Рис. 12.3. Основной экран приложения

Исходный код класса MainActivity и класса DBWork находится в прил. Е данного пособия.

Вопросы:

1. В каких случаях целесообразно использовать SharedPreferences, а в каких БД?
2. Преимущества и недостатки использования БД на мобильном устройстве.
3. Какой класс используется для открытия соединения с БД?

ЗАКЛЮЧЕНИЕ

В методических указаниях были рассмотрены основные подходы по программированию под мобильную ОС Android. Популярность ОС Android обусловлена широким использованием системы не только в мобильных 92

устройствах, но и в телевизорах, часах, автомобильных головных устройствах. При этом в каждом из устройств сохраняется единый подход по разработке программ, который и описан в учебном пособии.

Помимо стандартных библиотек для разработки под Android существует большое количество различных сторонних библиотек, реализующих тот или иной функционал. Такое разнообразие в первое время может вызывать трудности из-за большого выбора однотипных библиотек. Но постепенно это только облегчит разработку.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Чезарини, Ф. Программирование в Erlang / Ф. Чезарини, С. Томпсон. - М. : ДМК Пресс, 2012. - 487 с. - (Функциональное программирование). - ISBN 978-5-94074-617-1 ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=232035>
2. Операционная система Android / . - М. : МИФИ, 2012. - 64 с. - ISBN 978-5-7262-1780-2 ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=231690>
3. Михеева, Е. В. Информационные технологии в профессиональной деятельности : учеб.пособие / Е.В. Михеева. - 14-е изд., стер. - М. : Академия, 2016. - 384 с.
4. Гохберг, Г. С. Информационные технологии : учебник / Г.С. Гохберг, А.В. Зафиевский, А.А. Короткин. - 9-е изд., перераб. и доп. - М. : Академия, 2014. - 240 с.
5. Хлебников, А. А. Информационные технологии : учебник / А. А. Хлебников. – М. :КноРус, 2014. – 472 с.
6. Васильев, А. Н. Java. Объектно-ориентированное программирование : [учеб.пособие] / А.Н. Васильев. - СПб. : Питер, 2012. - 400 с. : ил. - (Учебное пособие). - Прил.: с. 379-395. - Библиогр.: с. 377.

7. Винокуров Н.А. Практика и теория программирования. В 2 Кн. Кн. 1 Ч. I и II [Текст]: учеб.издание/ Н.А. Винокуров, А.В. Ворожцов. – М.: Физматкнига, 2008. – 192 с.

8. Винокуров Н.А. Практика и теория программирования. В 2 Кн. Кн. 2 Ч. III и IV [Текст]: учеб.издание/ Н.А. Винокуров, А.В. Ворожцов. – М.: Физматкнига, 2008. – 288 с.

Приложение А Содержимое файла build.gradle

```
apply plugin: 'com.android.application' android
{ compileSdkVersion 23
buildToolsVersion "23.0.2" defaultConfig {
applicationId "ru.omgtu.myapplication" minSdkVersion 15
targetSdkVersion 23
versionCode 1
versionName "1.0"
}
buildTypes {
release {
minifyEnabledfalse
proguardFiles getDefaultProguardFile('proguard- android.txt'), 'proguard-rules.pro'
}
}
}
dependencies {
compile fileTree(dir: 'libs', include: ['*.jar']) testCompile
'junit:junit:4.12' compile 'com.android.support:appcompat-v7:23.2.1' }
```

Приложение Б Содержимое файла AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="ru.omgtu.myapplication">
    <application
        android:allowBackup="true"                android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name" android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Приложение В

Пример приложения для просмотра жизненного цикла Activity

Листинг activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"                android:layout_width="match_parent"
    android:layout_height="match_parent" android:orientation="vertical"
    tools:context=".MainActivity">
    </LinearLayout>
```

Листинг MainActivity.java

```
package ru.omgtu.myapplication;
import android.support.v7.app.ActionBarActivity; import
android.os.Bundle; import android.util.Log; import android.view.View;
public class MainActivity extends ActionBarActivity { private String TAG =
"MainActivity";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState); setContentView(R.layout.activity_main);
Log.i(TAG, "onCreate()");
    }
    @Override
    protected void onStart() { super.onStart(); Log.i(TAG, "onStart()");
    }
    @Override
    protected void onResume() { super.onResume(); Log.i(TAG, "onResume()");
    }
    @Override
    protected void onPause() { super.onPause(); Log.i(TAG, "onPause()");
```

```
}  
  
@Override  
protected void onStop() { super.onStop(); Log.i(TAG, "onStop()");  
}  
  
@Override  
protected void onRestart() { super.onRestart(); Log.i(TAG, "onRestart()");  
}  
  
@Override  
protected void onDestroy() { super.onDestroy(); Log.i(TAG, "onDestroy()");  
}  
}
```

Приложение Г

Пример приложения для переключения между экранами

Листинг MainActivity.java

```
package ru.omgtu.myapplication; import
android.content.Intent;

import android.support.v7.app.AppCompatActivity; import
android.os.Bundle; import android.view.View; import android.widget.Button;

public class MainActivity extends AppCompatActivity implements
View.OnClickListener{

    Button btnNextWindow; @Override

    protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState); setContentView(R.layout.activity_main);

        btnNextWindow =(Button) findViewById(R.id.btnNextWindow);
btnNextWindow.setOnClickListener(this);

    }

    @Override

    public void onClick(View v) { switch (v.getId())
    { case R.id.btnNextWindow:

        Intent intent = new Intent(this,NewActivity.class); startActivity(intent);
        break; default:

        break;

    }

    }

}
```

Листинг activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"                android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
```

```

android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context="ru.omgtu.myapplication.MainActivity">
    <Button
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:text="NextWindow" android:id="@+id/btnNextWindow" />
    </LinearLayout>

```

Листинг activity_new.xml

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="ru.omgtu.myapplication.NewActivity">
    <TextView
        android:layout_width="match_parent" android:layout_height="wrap_content"
        android:text="@string/txtAct2" android:textSize="50sp" />
    </RelativeLayout>

```


Приложение Д

Пример приложения для работы с SharedPreferences и файлами внутренней памяти

```
Листинг BaseActivity.java package ru.omgtu.myapplication; import
android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager; import
android.support.v4.app.FragmentTransaction; import
android.support.v7.app.AppCompatActivity; import com.squareup.otto.Bus;

public abstract class BaseActivity extends AppCompatActivity
{ private Bus bus; @Override
protected void onCreate(@Nullable Bundle savedInstanceState)
{ super.onCreate(savedInstanceState);
((AppCompatActivity).getApplication()).getBus(); }

@Override
protected void onStart() {
super.onStart();
register(this);
}

@Override
protected void onStop() {
super.onStop();
unregister(this);
}

@Override
protected void onDestroy() {
```

```

    super.onDestroy();
}

    public void replaceFragment(Fragment fragment) { replaceFragment(fragment,
true, null);
    }

    public void replaceFragment(Fragment fragment,
    boolean addToBackStack) {
        replaceFragment(fragment, addToBackStack, null);
    }

    public void replaceFragment(Fragment fragment, boolean
    addToBackStack, @Nullable String key) { FragmentTransaction
replaceTransaction =
        getSupportFragmentManager()
        .beginTransaction()
        .replace(R.id.container, fragment);
    if(addToBackStack) replaceTransaction
        .addToBackStack(key); replaceTransaction
        .commit();
    }

    public boolean returnToBackStack(String stackKey, boolean
    inclusive) {
        return getSupportFragmentManager()
        .popBackStackImmediate(stackKey, inclusive, ?
FragmentManager.POP_BACK_STACK_INCLUSIVE : 0);
    }

    public Bus getBus() {
        return bus;
    }
}

```

```

    Листинг BaseFragment.java package ru.omgtu.myapplication; import
android.os.Bundle;

    import android.support.annotation.Nullable; import
android.support.v4.app.Fragment; import android.view.View;

    import com.squareup.otto.Bus;
    import butterknife.ButterKnife;
    public class BaseFragment extends Fragment {
    private Bus bus; @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    }
    @Override
    public void onStart() {
    super.onStart();
    bus = ((BaseActivity) getActivity()).getBus();
    bus.register(this);
    }
    @Override
    public void onStop() { super.onStop(); bus.unregister(this);
    }
    @Override
    public void onDestroy() {
    super.onDestroy();
    }
    @Override
    public void onCreateView(View view, @Nullable Bundle savedInstanceState)
    { super.onViewCreated(view, savedInstanceState); ButterKnife.bind(this, view);

    }

```

@Override

```
public void onDestroyView() { super.onDestroyView(); ButterKnife.unbind(this);  
}
```

```
public BaseActivity getBaseActivity() {  
return (BaseActivity) getActivity();  
}
```

```
public Bus getBus() {  
return bus;  
}  
}
```

Листинг MainActivity.java **package** ru.omgtu.myapplication; **import**
android.content.Intent; **import** android.os.Bundle;

```
import com.squareup.otto.Subscribe;
```

```
public class MainActivity extends BaseActivity { @Override  
protected void onStop() {  
super.onStop();  
}
```

@Override

```
protected void onCreate(Bundle savedInstanceState) {  
super.onCreate(savedInstanceState); setContentView(R.layout.activity_main);  
if(savedInstanceState == null)
```

```
getSupportFragmentManager().beginTransaction().add(R.id.container,
```

```
new HW7MenuFragment()).commit(); }
```

@Subscribe

```
public void onOpenSaveInSharedPreferencesFragmentEvent(OpenSaveInSharedP  
referencesFragmentEvent event) {  
}
```

```

        replaceFragment(new SaveInSharedPreferencesFragment(), true);
        @Subscribe
        public void onOpenSaveInFileFragmentEvent(OpenSaveInFileFragmentEvent
event) {
            replaceFragment(new SaveInFileFragment(), true);
        }
    }
}

```

```

Листинг HW7MenuFragment.java package ru.omgtu.myapplication; import
android.os.Bundle;

import android.support.v4.app.Fragment; import android.view.LayoutInflater;
import android.view.View;

import android.view.ViewGroup; import butterknife.ButterKnife; import
butterknife.OnClick;

public class HW7MenuFragment extends BaseFragment
{ public HW7MenuFragment() {
    // Required empty public constructor
}

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_hw7_menu, container,
false); ButterKnife.bind(this, view);
        return view;
    }

    @Override
    public void onDestroyView() { super.onDestroyView(); ButterKnife.unbind(this);

}
}

```

```

    @OnClick(R.id.save_in_shared_preferences_button)
    public void onSaveInSharedPreferencesButtonClick() { getBus().post(new
    OpenSaveInSharedPreferencesFragmentEvent());
    }
    @OnClick(R.id.save_in_file_button)
    public void onSaveInFileButtonClick() { getBus().post(new
    OpenSaveInFileFragmentEvent());
    }
    @OnClick(R.id.phone_book_button)
    public void onPhoneBookButtonClick() { getBus().post(new
    OpenPhoneBookFragmentEvent());
    }
    }

```

Листинг SaveInSharedPreferencesFragment.java

```

package ru.omgtu.myapplication;
import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.v4.app.Fragment; import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup; import android.widget.EditText; import
butterknife.Bind;
import butterknife.ButterKnife;
public class SaveInSharedPreferencesFragment
extends BaseFragment {
    public static final String APP_PREFERENCES = "myPreferences"; public
static final String APP_PREFERENCES_TEXT = "Text"; SharedPreferences
sharedPreferences; @Bind(R.id.edit_text_for_save)
    EditText editTextForSave;

```

```

public SaveInSharedPreferencesFragment()
{ // Required empty public constructor
}

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_save_text, container,
false); ButterKnife.bind(this, view);

    sharedPreferences= getContext().getSharedPreferences(APP_PREFERENCES,
Context.MODE_PRIVATE);

    return view;
}

@Override

public void onDestroyView() { super.onDestroyView(); ButterKnife.unbind(this);
}

@Override

public void onPause() {
    super.onPause();
    SharedPreferences.Editor editor = sharedPreferences.edit();
    editor.putString(APP_PREFERENCES_TEXT,
        editTextForSave.getText().toString()); editor.apply();
}

@Override

public void onResume() {
    super.onResume();

    if(sharedPreferences.contains(APP_PREFERENCES_TEXT)) {
        editTextForSave.setText(sharedPreferences.getString(APP_PREFERENCES_TEXT,
            ""));
    }
}

```

```

    }

    Пустинг SaveInFileFragment.java package ru.omgtu.myapplication; import
android.content.Context; import android.os.Bundle;

import android.os.Environment;
import android.support.v4.app.Fragment;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View; import android.view.ViewGroup; import
android.widget.EditText; import android.widget.Toast; import
ru.omgtu.myapplication.R; import java.io.BufferedReader; import java.io.File;
import java.io.FileInputStream; import java.io.FileNotFoundException; import
java.io.FileOutputStream; import java.io.InputStream;
import java.io.InputStreamReader; import java.io.OutputStream; import
java.io.OutputStreamWriter; import butterknife.Bind;
import butterknife.ButterKnife;

public class SaveInFileFragment extends BaseFragment { public static final
String FILE_NAME = "fileForSaveText.txt"; File file;

    @Bind(R.id.edit_text_for_save) EditText editTextForSave; public
SaveInFileFragment() {
    // Required empty public constructor
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {
    // Inflate the layout for this fragment
    View view = inflater.inflate(R.layout.fragment_save_text, container,
false); ButterKnife.bind(this, view);
    return view;
    }

    @Override

```



```

public void onDestroyView() { super.onDestroyView(); ButterKnife.unbind(this);
}

@Override
public void onPause() {
super.onPause(); try {
    OutputStream outputStream = getContext().openFileOutput(FILE_NAME,
Context.MODE_PRIVATE);
    OutputStreamWriter osw = new
    OutputStreamWriter(outputStream);
    osw.write(editTextForSave.getText().toString()); osw.close();
    outputStream.close();
} catch (Throwable t) { Toast.makeText(getContext(),
"Exception: "+ t.toString(),
Toast.LENGTH_LONG).show(); }
}

@Override
public void onResume() {
super.onResume(); try {
    InputStream inputStream = getContext().openFileInput(FILE_NAME);
    InputStreamReader isr = new InputStreamReader(inputStream); BufferedReader
reader = new BufferedReader(isr); editTextForSave.setText(reader.readLine());
    reader.close(); isr.close(); inputStream.close();
} catch (Throwable t) { Toast.makeText(getContext(),
"Exception: "+ t.toString(),
Toast.LENGTH_LONG).show(); }
}
}

```

Приложение Е

Пример приложения «Телефонная книга»

Листинг activity_main.xml

```
<EditText
    android:id="@+id/etName"                                android:hint="Name"
    android:layout_width="match_parent" android:layout_height="wrap_content">
</EditText>
<EditText android:id="@+id/etEmail" android:hint="Email"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
</EditText>
<LinearLayout android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <Button android:id="@+id/btnAdd"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Add">
    </Button>
    <Button android:id="@+id/btnRead"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Read">
    </Button>
    <Button android:id="@+id/btnClear"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Clear">
    </Button>
```

```

Пустинг MainActivity.java package ru.omgtu.myapplication; import
android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log; import android.view.View; import
android.widget.Button;
import android.widget.EditText;
import android.content.ContentValues;
public class MainActivity extends AppCompatActivity implements
View.OnClickListener{
Button btnAdd, btnRead, btnClear; EditText etName, etNumb;
DBWork dbWork; @Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState); setContentView(R.layout.activity_main);
btnAdd= (Button) findViewById(R.id.btnRecord);
btnAdd.setOnClickListener(this);
btnRead= (Button) findViewById(R.id.btnReading);
btnRead.setOnClickListener(this);
btnClear= (Button) findViewById(R.id.btnRemoval);
btnClear.setOnClickListener(this);
etName= (EditText) findViewById(R.id.etName); etNumb = (EditText)
findViewById(R.id.etNumb); dbWork = new DBWork(this);
}
@Override
public void onClick(View v) {
String name = etName.getText().toString(); String numb =
etNumb.getText().toString();
SQLiteDatabase database = dbWork.getWritableDatabase();

```

```

ContentValues contentValues = new
ContentValues(); switch(v.getId()) {
    case R.id.btnRecord: contentValues.put(DBWork.KEY_NAME, name);
contentValues.put(DBWork.KEY_NUMBER, numb);
database.insert(DBWork.TABLE_CONTACTS, null,
    contentValues);
break;
    case R.id.btnReading: Cursor cursor =
database.query(DBWork.TABLE_CONTACTS, null, null, null, null, null, null);
if(cursor.moveToFirst()) {
    intidIndex = cursor.getColumnIndex(DBWork.KEY_ID);
    intnameIndex = cursor.getColumnIndex(DBWork.KEY_NAME);
    intnumbIndex = cursor.getColumnIndex(DBWork.KEY_NUMBER);
    do{
        Log.d("mLog", "ID = " + cursor.getInt(idIndex) + ", name = " +
cursor.getString(nameIndex) + ", numb = " + cursor.getString(numbIndex));
    } while
(cursor.moveToNext()); } else
Log.d("mLog", "0 rows"); cursor.close();
break;
    case R.id.btnRemoval: database.delete(DBWork.TABLE_CONTACTS, null, null);
break;
}
dbWork.close();
}
}

```

```

Листинг DBWork.java package ru.omgtu.myapplication; import
android.content.Context;

import android.database.sqlite.SQLiteDatabase; import
android.database.sqlite.SQLiteOpenHelper; public class DBWork extends
SQLiteOpenHelper {

    public static final int DATABASE_VERSION = 1;

    public static final String DATABASE_NAME = "phoneDb"; public static final
String TABLE_CONTACTS = "contacts"; public static final String KEY_ID = "_id";

    public static final String KEY_NAME = "name"; public static final String
KEY_NUMBER = "number"; public DBWork(Context context) {
    super(context, DATABASE_NAME, null,
DATABASE_VERSION);
    }

    @Override

    public void onCreate(SQLiteDatabase db) { db.execSQL("create table " +
TABLE_CONTACTS + "(" +
KEY_ID
+ " integer primary key," + KEY_NAME + " text,"
+ KEY_NUMBER + " text" + ")"); }

    @Override

    public void onUpgrade(SQLiteDatabase db, int oldVersion,
int newVersion) {
    db.execSQL("drop table if exists " + TABLE_CONTACTS); onCreate(db);
    }
}

```