

```

transaction.replace(R.id.fragment_single, fragInfo);
transaction.commit();

Фрагмент:

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState)
{
    String myValue = this.getArguments().getString("message");
    ...
}

```

## 42.2. Шаблон newInstance()

Можно создать конструктор фрагмента с параметрами, но Android внутренне вызывает конструктор с нулевым аргументом при пересоздании фрагментов (например, если они восстанавливаются после уничтожения по собственным причинам Android). Поэтому полагаться на конструктор с параметрами не рекомендуется.

Чтобы гарантировать, что ожидаемые аргументы фрагмента всегда будут присутствовать, можно использовать статический метод newInstance() для создания фрагмента и поместить все необходимые параметры в объект Bundle, который будет доступен при создании нового экземпляра.

```

import android.os.Bundle;
import android.support.v4.app.Fragment;

public class MyFragment extends Fragment
{
    // Наш идентификатор для получения имени из аргументов
    private static final String NAME_ARG = "имя";
    private String mName;

    // Требуется
    public MyFragment(){}
}

// Статический конструктор. Это единственный способ инициализировать фрагмент
public static MyFragment newInstance(final String name) {
    final MyFragment myFragment = new MyFragment();
    // Приведенная ниже цифра 1 является оптимизацией – количеством аргументов,
    // которые будут добавлены в этот бандл.
    // Если известно количество аргументов, которые будут добавлены в бандл,
    // то прекращается дополнительное выделение карты поддержки (Backing Map).
    // Если вы не уверены, можно сконструировать бандл без аргументов
    final Bundle args = new Bundle(1);

    // При этом аргумент сохраняется как аргумент в бандле. Обратите внимание,
    // что даже если параметр 'name' имеет значение NULL,
    // то это будет работать, поэтому следует в этот момент проверить,
    // является ли параметр обязательным, и если да, то проверить на наличие NULL
    // и в этом случае выдать соответствующую ошибку
    args.putString(NAME_ARG, name);

    myFragment.setArguments(args);
    return myFragment;
}

@Override
public void onCreate(Bundle savedInstanceState) {

```

```
super.onCreate(savedInstanceState);
final Bundle arguments = getArguments();
if (arguments == null || !arguments.containsKey(NAME_ARG)) {
    // Задайте значение по умолчанию или ошибку по своему усмотрению
} else {
    mName = arguments.getString(NAME_ARG);
}
}
```

Теперь в активности:

```
FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
MyFragment mFragment = MyFragment.newInstance("мое имя");
ft.replace(R.id.placeholder, mFragment);
//R.id.placeholder - это место, куда мы хотим загрузить наш фрагмент
ft.commit();
```

Этот паттерн – лучшая практика, гарантирующая, что все необходимые аргументы будут переданы фрагментам при создании. Обратите внимание, что при уничтожении фрагмента и последующем его воссоздании система автоматически восстановит его состояние, но для этого необходимо обеспечить реализацию `onSaveInstanceState(Bundle)`.

## 42.3. Навигация между фрагментами с помощью обратного стека и шаблона Static Fabric

Прежде всего нам необходимо добавить наш первый фрагмент в начало, сделать это нужно в методе `onCreate()` нашей активности:

```
if (null == savedInstanceState) {  
    getSupportFragmentManager().beginTransaction()  
        .addToBackStack("fragmentA")  
        .replace(R.id.container, FragmentA.newInstance(), "fragmentA")  
        .commit();  
}
```

Далее нам необходимо управлять нашим обратным стеком. Проще всего это сделать с помощью функции, добавленной в нашу активность, которая используется для всех операций с фрагментами.

```
public void replaceFragment(Fragment fragment, String tag) {  
    //Получение текущего фрагмента, помещенного в контейнер  
    Fragment currentFragment = getSupportFragmentManager().findFragmentById(R.  
        id.container);  
  
    //Предотвращение добавления одного и того же фрагмента сверху  
    if (currentFragment.getClass() == fragment.getClass()) {  
        return;  
    }  
  
    //Если фрагмент уже находится в стеке, мы можем вернуть (pop back) стек назад,  
    //чтобы предотвратить бесконечный рост  
    if (getSupportFragmentManager().findFragmentByTag(tag) != null) {  
        getSupportFragmentManager().popBackStack(tag, FragmentManager.POP_BACK_STACK_  
            INCLUSIVE);  
    }  
  
    //В противном случае просто заменим фрагмент
```

```

getSupportFragmentManager()
    .beginTransaction()
    .addToBackStack(tag)
    .replace(R.id.container, fragment, tag)
    .commit();
}

```

Наконец, следует переопределить `onBackPressed()`, чтобы завершить работу приложения при возврате с последнего фрагмента, доступного в обратном стеке.

```

@Override
public void onBackPressed() {
    int fragmentsInStack = getSupportFragmentManager().getBackStackEntryCount();
    if (fragmentsInStack > 1) { // Если у нас больше одного фрагмента, возвращаем
        стек назад
        getSupportFragmentManager().popBackStack();
    } else if (fragmentsInStack == 1) { // Завершаем активность, если остался только
        один фрагмент, чтобы не оставлять пустой экран
        finish();
    } else {
        super.onBackPressed();
    }
}

```

Выполнение в активности:

```
replaceFragment(FragmentB.newInstance(), "fragmentB");
```

Выполнение вне активности (предполагается, что наша активность – это `MainActivity`):

```
((MainActivity) getActivity()).replaceFragment(FragmentB.newInstance(), "fragmentB");
```

## 42.4. Отправка событий обратно в активность с помощью интерфейса обратного вызова

Если необходимо передавать события от фрагмента к активности, то одно из возможных решений – определить интерфейс обратного вызова и потребовать от принимающей активности реализовать его.

### Пример

Отправим обратный вызов в активность при нажатии на кнопку фрагмента.

Прежде всего определите интерфейс обратного вызова:

```

public interface SampleCallback {
    void onButtonClicked();
}

```

Следующий шаг – назначение этого обратного вызова во фрагменте:

```

public final class SampleFragment extends Fragment {

    private SampleCallback callback;

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        if (context instanceof SampleCallback) {

```

```

        callback = (SampleCallback) context;
    } else {
        throw new RuntimeException(context.toString()
                + " must implement SampleCallback");
    }
}

@Override
public void onDetach() {
    super.onDetach();
    callback = null;
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
    final View view = inflater.inflate(R.layout.sample, container, false);
    // Добавление слушателя нажатия кнопки
    view.findViewById(R.id.actionButton).setOnClickListener(new View.
    OnClickListener() {
        public void onClick(View v) {
            callback.onButtonClicked(); // здесь обратный вызов
        }
    });
    return view;
}
}

```

И, наконец, реализуем обратный вызов в активности:

```

public final class SampleActivity extends Activity implements SampleCallback {

// ...Пропущен код с настройками представления содержимого и представлением фрагмента

    @Override
    public void onButtonClicked() {
        // Вызывается при нажатии на кнопку фрагмента
    }
}

```

## 42.5. Анимирование перехода между фрагментами

Чтобы анимировать переход между фрагментами или процесс показа либо скрытия фрагмента, необходимо с помощью FragmentManager создать FragmentTransaction.

Для одной транзакции FragmentTransaction существует два различных пути: можно использовать стандартную анимацию, а можно предоставить собственную пользовательскую анимацию.

Стандартные анимации задаются вызовом FragmentTransaction.setTransition(int transit) и использованием одной из предопределенных констант, имеющихся в классе FragmentTransaction. Среди них:

```

FragmentTransaction.TRANSIT_NONE
FragmentTransaction.TRANSIT_FRAGMENT_OPEN
FragmentTransaction.TRANSIT_FRAGMENT_CLOSE
FragmentTransaction.TRANSIT_FRAGMENT_FADE

```

Полная транзакция может выглядеть следующим образом:

```
getSupportFragmentManager()
    .beginTransaction()
    .setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE)
    .replace(R.id.contents, new MyFragment(), "MyFragmentTag")
    .commit();
```

Пользовательские анимации допускают два способа реализации: задаются вызовом – либо `FragmentTransaction.setCustomAnimations(int enter, int exit)`, либо `FragmentTransaction.setCustomAnimations(int enter, int exit, int popEnter, int popExit)`.

Анимация входа и выхода будет воспроизводиться для операций `FragmentTransactions`, не связанных с выгрузкой фрагментов из обратного стека. Анимации `popEnter` и `popExit` будут воспроизводиться при выгрузке фрагмента из обратного стека.

Следующий код показывает, как можно заменить фрагмент, выдвигая (*sliding out*) один фрагмент и вставляя на его место другой.

```
getSupportFragmentManager()
    .beginTransaction()
    .setCustomAnimations(R.anim.slide_in_left, R.anim.slide_out_right)
    .replace(R.id.contents, new MyFragment(), "MyFragmentTag")
    .commit();
```

В XML-определениях анимации будет использоваться тег `objectAnimator`. Пример файла `slide_in_left.xml` может выглядеть следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<set>
    <objectAnimator xmlns:android="http://schemas.android.com/apk/res/android"
        android:propertyName="x"
        android:valueType="floatType"
        android:valueFrom="-1280"
        android:valueTo="0"
        android:duration="500" />
</set>
```

## 42.6. Связь между фрагментами

Все коммуникации между фрагментами должны осуществляться через активность. Фрагменты **НЕ МОГУТ** сообщаться между собой без активности.

### Дополнительные ресурсы

- Как реализовать `OnFragmentInteractionListener`: <https://stackoverflow.com/questions/24777985/how-to-implement-onfragmentinteractionlistener>
- Android | связь между фрагментами: <https://developer.android.com/training/basics/fragments/communicating.html>

В данном примере мы имеем `MainActivity`, в которой размещены два фрагмента, `SenderFragment` и `ReceiverFragment`, для отправки и приема сообщения (в данном случае просто `string`) соответственно.

Кнопка во фрагменте `SenderFragment` инициирует процесс отправки сообщения. При получении сообщения обновляется `TextView` во фрагменте `ReceiverFragment`.

Ниже приведен фрагмент для `MainActivity` с комментариями, поясняющими важные строки кода:

```
// Наша MainActivity реализует интерфейс, определенный фрагментом SenderFragment.
// Это позволяет осуществить передачу от фрагмента к активности
public class MainActivity extends AppCompatActivity implements SenderFragment.
SendMouseListener {
```

```

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

/**
 * Этот метод вызывается, когда мы нажимаем на кнопку во фрагменте SenderFragment
 * @param message - сообщение, отправленное фрагментом SenderFragment
 */
@Override
public void onSendMessage(String message) {
    // Находим наш ReceiverFragment, используя SupportFragmentManager и id фрагмента
    ReceiverFragment receiverFragment = (ReceiverFragment)
        getSupportFragmentManager().findFragmentById(R.id.fragment_receiver);

    // Убеждаемся, что такой фрагмент существует
    if (receiverFragment != null) {
        // Отправляем это сообщение фрагменту ReceiverFragment, вызвав его открытый метод
        receiverFragment.showMessage(message);
    }
}
}

```

В файле компоновки для MainActivity внутри LinearLayout размещены два фрагмента :

```

<?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:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    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="com.naru.fragmentcommunication.MainActivity">

    <fragment
        android:id="@+id/fragment_sender"
        android:name="com.naru.fragmentcommunication.SenderFragment"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        tools:layout="@layout/fragment_sender" />

    <fragment
        android:id="@+id/fragment_receiver"
        android:name="com.naru.fragmentcommunication.ReceiverFragment"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        tools:layout="@layout/fragment_receiver" />
</LinearLayout>

```

Фрагмент SenderFragment раскрывает интерфейс SendMessageListener, с помощью которого MainActivity узнает, когда была нажата кнопка в фрагменте SenderFragment.

Ниже приведен фрагмент кода для SenderFragment с пояснением важных строк:

```
public class SenderFragment extends Fragment {  
  
    private SendMessageListener commander;  
  
    /**  
     * Этот интерфейс предназначен для связи между активностью и фрагментом.  
     * Любая активность, реализующая этот интерфейс, сможет  
     * получить сообщение, передаваемое этим фрагментом.  
     */  
    public interface SendMessageListener {  
        void onSendMessage(String message);  
    }  
  
    /**  
     * УРОВЕНЬ API >= 23  
     * Этот метод вызывается, когда фрагмент присоединяется к активности. В данном  
     * случае этот метод поможет нам инициализировать  
     * ссылочную переменную 'commander' для нашего интерфейса 'SendMessageListener'  
     *  
     * @param context  
     */  
    @Override  
    public void onAttach(Context context) {  
        super.onAttach(context);  
        // Попытаемся привести контекст к нашему интерфейсу SendMessageListener,  
        // то есть проверим, реализует ли активность SendMessageListener.  
        // В противном случае возникает исключение ClassCastException.  
        try {  
            commander = (SendMessageListener) context;  
        } catch (ClassCastException e) {  
            throw new ClassCastException(context.toString()  
                + " must implement the SendMessageListener interface");  
        }  
    }  
  
    @Nullable  
    @Override  
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container,  
                            @Nullable Bundle savedInstanceState) {  
        // "Раздуваем" представление для фрагмента-отправителя.  
        View view = inflater.inflate(R.layout.fragment_receiver, container, false);  
  
        // Инициализация кнопки и слушателя щелчка на ней  
        Button send = (Button) view.findViewById(R.id.bSend);  
        send.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View v) {  
                // Проверка правильности инициализации ссылки на интерфейс  
                if (commander != null) {  
                    // Вызов нашего метода интерфейса. Это позволяет нам вызвать  
                    // реализованный метод в активности, откуда мы можем отправить сообщение  
                    // в ReceiverFragment.  
                    commander.onSendMessage("HELLO FROM SENDER FRAGMENT!");  
                }  
            }  
        });  
        return view;  
    }  
}
```

```

    }
});

return view;
}
}

```

Файл компоновки для фрагмента SenderFragment:

```

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

    <Button
        android:id="@+id/bSend"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="SEND"
        android:layout_gravity="center_horizontal" />
</LinearLayout>

```

Фрагмент ReceiverFragment прост и предоставляет несложный открытый метод для обновления своего TextView. Когда MainActivity получает сообщение от фрагмента SenderFragment, она вызывает этот открытый метод фрагмента ReceiverFragment.

Ниже приведен код для ReceiverFragment с комментариями, поясняющими важные строки:

```

public class ReceiverFragment extends Fragment {
    TextView tvMessage;

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container,
                            @Nullable Bundle savedInstanceState) {
        // "Раздуваем" представление для фрагмента-отправителя.
        View view = inflater.inflate(R.layout.fragment_receiver, container, false);

        // Инициализация TextView
        tvMessage = (TextView) view.findViewById(R.id.tvReceivedMessage);

        return view;
    }

    /**
     * Метод, вызываемый MainActivity при получении сообщения от SenderFragment.
     * Этот метод позволяет обновить текст в TextView в соответствии с сообщением,
     * отправленным SenderFragment.
     * @param message - сообщение, отправленное фрагментом SenderFragment через
     * MainActivity.
     */
    public void showMessage(String message) {
        tvMessage.setText(message);
    }
}

```

Файл компоновки для ReceiverFragment:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical">
    <TextView
        android:id="@+id/tvReceivedMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Waiting for message!" />
</LinearLayout>
```

## Глава 43. Кнопка

### 43.1. Использование одного и того же события щелчка для одного или нескольких представлений в XML

При создании любого представления в макете мы можем использовать атрибут `android:onClick` для ссылки на метод в связанной с ним активности или фрагменте для обработки событий щелчка (прикосновения, клика).

#### Макет XML

```
<Button android:id="@+id/button"
    ...
    // onClick должен ссылаться на метод в вашей активности или фрагменте
    android:onClick="doSomething" />

// Обратите внимание, что это работает с любым классом, который является
// подклассом View, а не только с Button
<ImageView android:id="@+id/image"
    ...
    android:onClick="doSomething" />
```

#### Код активности/фрагмента

В своем коде создайте названный метод, где `v` будет представлением, которого коснулись, и сделайте что-нибудь для каждого представления, вызывающего этот метод.

```
public void doSomething(View v) {
    switch(v.getId()) {
        case R.id.button:
            // Кнопка была нажата, сделайте что-нибудь.
            break;
        case R.id.image:
            // Изображение было нажато, сделайте что-нибудь еще.
            break;
    }
}
```

При желании для каждого `View` можно использовать свой метод (в этом случае, конечно, не нужно проверять наличие ID).

## 43.2. Определение внешнего слушателя

Когда это следует использовать

- В случае если код внутри встроенного слушателя слишком велик, и ваш метод или класс становится некрасивым и трудночитаемым.
- Вы хотите выполнять одно и то же действие в различных элементах (представлениях) вашего приложения.

Для этого необходимо создать класс, реализующий один из слушателей API View.

Например, для подсказки при длительном нажатии на любой элемент:

```
public class HelpLongClickListener implements View.OnLongClickListener {
    public HelpLongClickListener() {
    }
    @Override
    public void onLongClick(View v) {
        // показать тест или всплывающее окно помощи
    }
}
```

Для его использования достаточно иметь в активности атрибут или переменную:

```
HelpLongClickListener helpListener = new HelpLongClickListener(...);

button1.setOnClickListener(helpListener);
button2.setOnClickListener(helpListener);
label.setOnClickListener(helpListener);
button1.setOnClickListener(helpListener);
```

**Примечание:** определение слушателей в выделенном классе имеет один недостаток – они не могут получить прямой доступ к полям класса, поэтому необходимо передавать данные (контекст, представление) через конструктор, если не делать атрибуты открытыми или не определить геттеры.

## 43.3. Встроенный слушатель

Допустим, у нас есть кнопка (мы можем создать ее программно или привязать из представления с помощью функции `findViewById()` и т.п.).

```
Button btnOK = (...)
```

Теперь создадим анонимный класс и встроим его:

```
btnOk.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Выполнить здесь некоторый код...
    }
});
```

## 43.4. Настройка стиля кнопок

Существует множество вариантов настройки внешнего вида кнопок. В данном примере представлено несколько из них:

**Вариант 0: Использовать ThemeOverlay (в настоящее время это самый простой и быстрый способ)**

Создайте новый стиль в файле стилей:

### styles.xml

```
<resources>
    <style name="mybutton" parent="ThemeOverlay.AppCompat.Light">
        <!-- настройка colorButtonNormal для цвета отключения -->
        <item name="colorButtonNormal">@color/colorbuttonnormal</item>
        <!-- настройка colorAccent для включенного цвета -->
        <item name="colorButtonNormal">@color/coloraccent</item>
    </style>
</resources>
```

Затем в макете, где вы размещаете свою кнопку (например, MainActivity):

### 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:layout_gravity="center_horizontal"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    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=".MainActivity">

    <Button
        android:id="@+id/mybutton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello"
        android:theme="@style/mybutton"
        style="@style/Widget.AppCompat.Button.Colored"/>
</LinearLayout>
```

**Вариант 1: Создание собственного стиля кнопок**

В файле values/styles.xml создайте новый стиль для кнопки:

### styles.xml

```
<resources>
    <style name="mybuttonstyle" parent="@android:style/Widget.Button">
        <item name="android:gravity">center_vertical|center_horizontal</item>
        <item name="android:textColor">#FFFFFFFF</item>
        <item name="android:shadowColor">#FF000000</item>
        <item name="android:shadowDx">0</item>
        <item name="android:shadowDy">-1</item>
    </style>
</resources>
```

```

<item name="android:shadowRadius">0.2</item>
<item name="android:textSize">16dp</item>
<item name="android:textStyle">bold</item>
<item name="android:background">@drawable/button</item>
</style>
</resources>

```

Затем в макете, где вы размещаете свою кнопку (например в MainActivity):

### 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:layout_gravity="center_horizontal"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    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=".MainActivity">
    <Button
        android:id="@+id/mybutton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello"
        android:theme="@style/mybuttonstyle"/>
</LinearLayout>

```

### Вариант 2: Назначить drawable-ресурс для каждого состояния кнопки

Создайте xml-файл с именем 'mybuttondrawable.xml' в папке drawable для определения drawable-ресурса для каждого состояния кнопки:

### drawable/mybutton.xml

```

<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:state_enabled="false"
        android:drawable="@drawable/mybutton_disabled" />
    <item
        android:state_pressed="true"
        android:state_enabled="true"
        android:drawable="@drawable/mybutton_pressed" />
    <item
        android:state_focused="true"
        android:state_enabled="true"
        android:drawable="@drawable/mybutton_focused" />
    <item
        android:state_enabled="true"
        android:drawable="@drawable/mybutton_enabled" />
</selector>

```

Каждый из этих drawable-ресурсов может быть изображением (например mybutton\_disabled.png) или xml-файлом, заданным вами и хранящимся в папке drawables. Например:

**drawable/mybutton\_disabled.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="#F2F2F2"
        android:centerColor="#A4A4A4"
        android:endColor="#F2F2F2"
        android:angle="90"/>
    <padding
        android:left="7dp"
        android:top="7dp"
        android:right="7dp"
        android:bottom="7dp" />
    <stroke
        android:width="2dp"
        android:color="#FFFFFF" />
    <corners
        android:radius="8dp" />
</shape>
```

Затем в макете, где вы размещаете свою кнопку (например `MainActivity`):

**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:layout_gravity="center_horizontal"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    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=".MainActivity">

    <Button
        android:id="@+id/mybutton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello"
        android:background="@drawable/mybuttondisabled" />
</LinearLayout>
```

**Вариант 3: Добавить свой стиль кнопок в тему приложения**

Вы можете переопределить стандартный стиль кнопок Android в определении темы приложения (в `values/styles.xml`):

**styles.xml**

```
<resources>
    <style name="AppTheme" parent="android:Theme">
```

```

<item name="colorPrimary">@color/colorPrimary</item>
<item name="colorPrimaryDark">@color/colorPrimaryDark</item>
<item name="colorAccent">@color/colorAccent</item>
<item name="android:button">@style/mybutton</item>

</style>
<style name="mybutton" parent="android:style/Widget.Button">
    <item name="android:gravity">center_vertical|center_horizontal</item>
        <item name="android:textColor">#FFFFFF</item>
        <item name="android:shadowColor">#FF000000</item>
        <item name="android:shadowDx">0</item>
        <item name="android:shadowDy">-1</item>
        <item name="android:shadowRadius">0.2</item>
        <item name="android:textSize">16dip</item>
        <item name="android:textStyle">bold</item>
        <item name="android:background">@drawable/anydrawable</item>
    </style>
</resources>

```

#### **Вариант 4: Программное наложение цвета на стиль кнопки по умолчанию**

Просто найдите кнопку в своей активности и примените цветовой фильтр:

```

Button mybutton = (Button) findViewById(R.id.mybutton);
mybutton.getBackground().setColorFilter(anycolor, PorterDuff.Mode.MULTIPLY)

```

Различные режимы смешивания можно посмотреть здесь: <https://developer.android.com/reference/android/graphics/PorterDuff.Mode.html>, а красивые примеры – здесь: <http://sspi.motussoft.com/porterduff.html>.

### **43.5. Пользовательский ClickListener для предотвращения многократных быстрых щелчков**

Чтобы предотвратить многократное срабатывание кнопки в течение короткого промежутка времени (скажем, 2 нажатия в течение 1 секунды, что может привести к серьезным проблемам, если поток не контролируется), можно реализовать пользовательский `SingleClickListener`.

Этот `ClickListener` задает в качестве порога определенный временной интервал (например, 1000 мс). При нажатии на кнопку будет выполнена проверка, выполнялся ли триггер в течение заданного вами времени, и если нет, то он сработает.

```

public class SingleClickListener implements View.OnClickListener {

    protected int defaultInterval;
    private long lastTimeClicked = 0;

    public SingleClickListener() {
        this(1000);
    }

    public SingleClickListener(int minInterval) {
        this.defaultInterval = minInterval;
    }

    @Override
    public void onClick(View v) {

```

### 43.6. Использование макета для определения действия щелчка

```

if (SystemClock.elapsedRealtime() - lastTimeClicked < defaultInterval) {
    return;
}
lastTimeClicked = SystemClock.elapsedRealtime();
performClick(v);
}

public abstract void performClick(View v);

}

```

А в классе к рассматриваемой кнопке привязан SingleClickListener:

```

myButton = (Button) findViewById(R.id.my_button);
myButton.setOnClickListener(new SingleClickListener() {
    @Override
    public void performClick(View view) {
        // выполнять что-то
    }
});

```

## 43.6. Использование макета для определения действия щелчка

Когда мы создаем кнопку в макете, мы можем использовать атрибут `android:onClick` для ссылки на метод в коде для обработки щелчков:

### Button

```

<Button
    android:width="120dp"
    android:height="wrap_content"
    android:text="Нажмите на меня"
    android:onClick="handleClick" />

```

Затем в своей активности создайте метод `handleClick`:

```

public void handleClick(View v) {
    // Делайте что угодно
}

```

## 43.7. Прослушивание событий длинного щелчка

Чтобы перехватить длинный щелчок (long click) и использовать его, необходимо предоставить кнопке соответствующий слушатель:

```

View.OnLongClickListener listener = new View.OnLongClickListener() {
    public boolean onLongClick(View v) {
        Button clickedButton = (Button) v;
        String buttonText = clickedButton.getText().toString();
        Log.v(TAG, "кнопка нажата продолжительно --> " + buttonText);
        return true;
    }
};

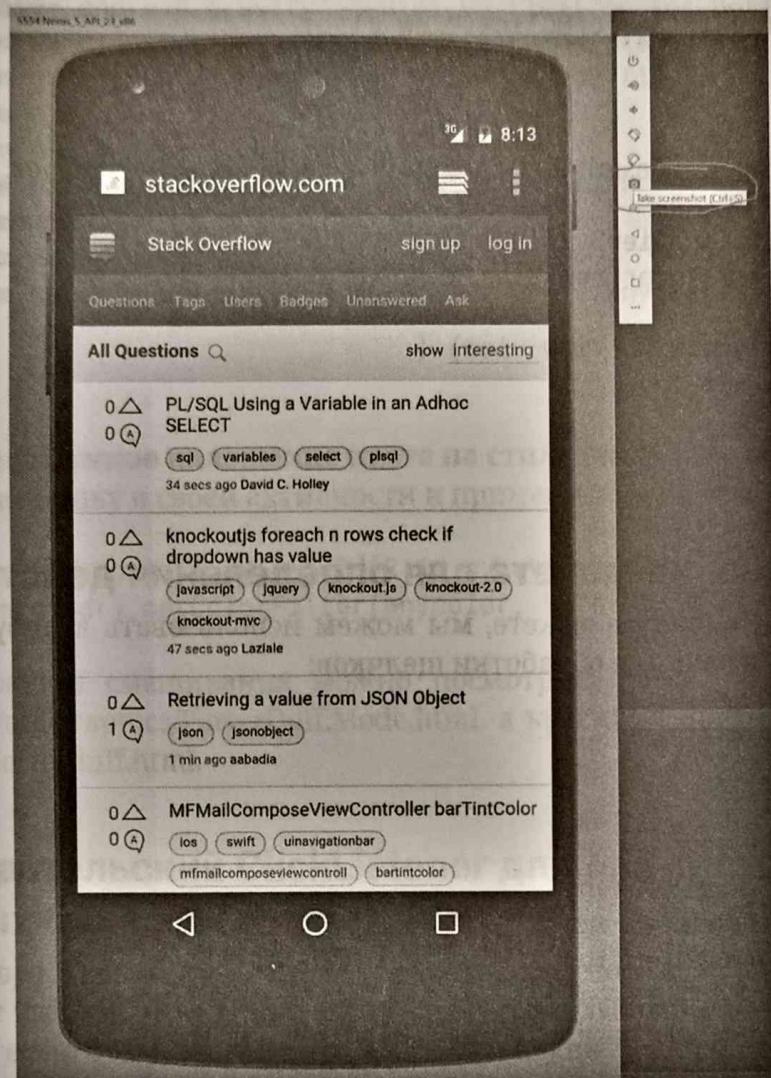
button.setOnLongClickListener(listener);

```

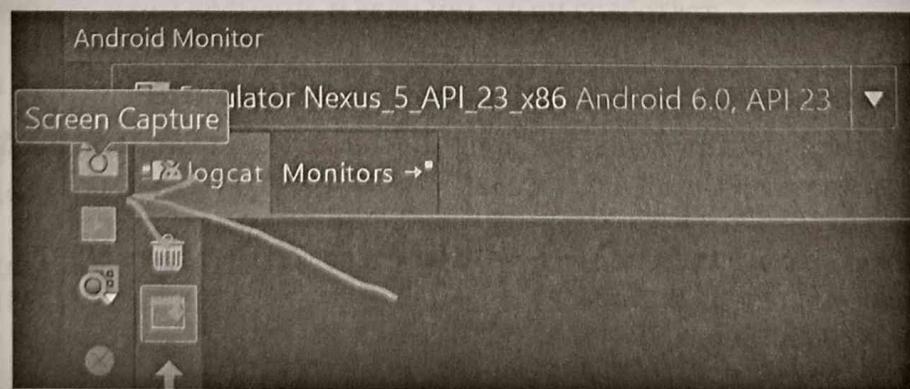
# Глава 44. Эмулятор

## 44.1. Создание скриншотов

Если вы хотите сделать снимок экрана из эмулятора Android Emulator, то можно выполнить команду Ctrl + S или нажать на значок камеры в боковой панели:



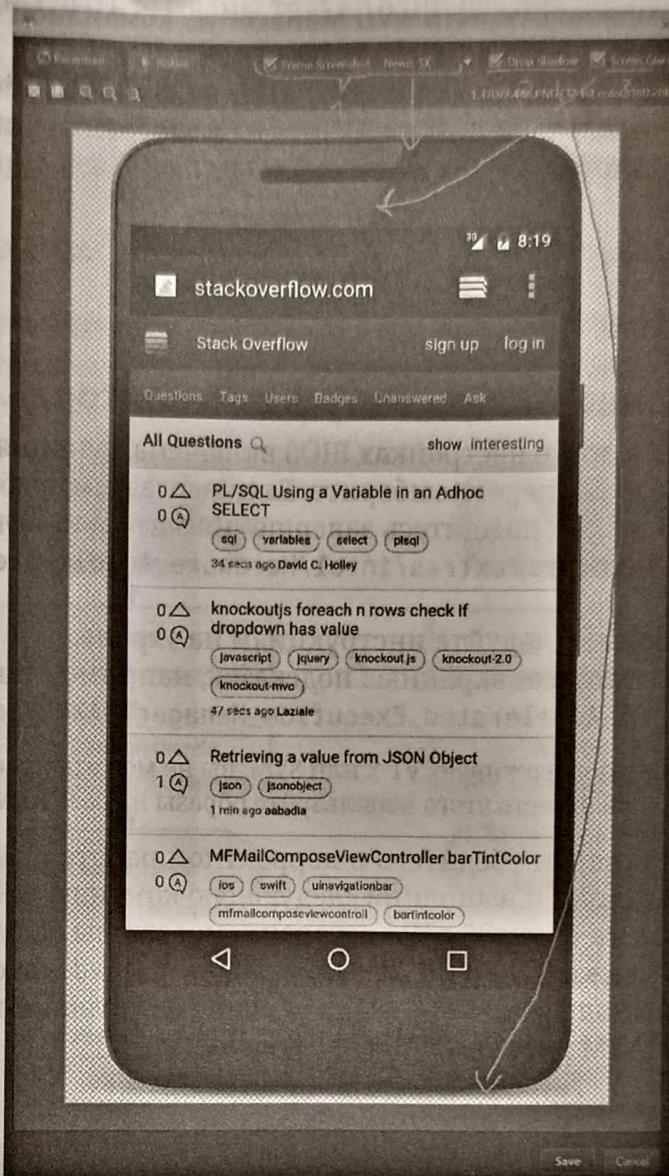
Если вы используете старую версию эмулятора Android или хотите сделать снимок экрана с реального устройства, то необходимо нажать на значок камеры в Android Monitor:



Дважды проверьте правильность выбора устройства, так как это является распространенной ошибкой. Сделав снимок экрана, можно по желанию добавить к нему следующие украшения (см. также рисунок ниже).

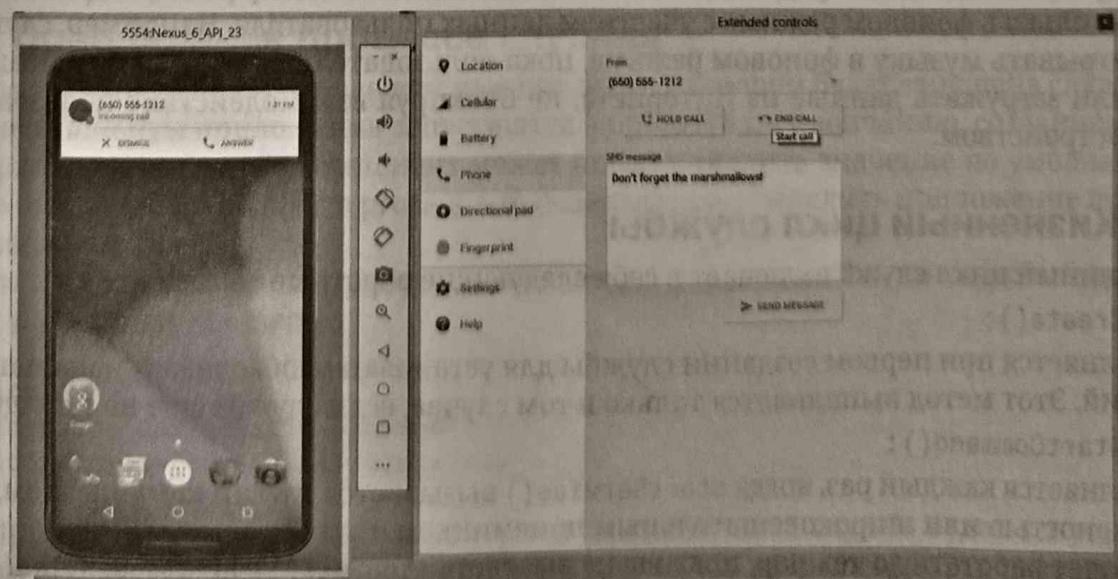
1. Рамка устройства вокруг снимка экрана.
2. Падающая тень.

### 3. Блики на экране.



## 44.2. Имитация вызова

Для имитации телефонного звонка нажмите кнопку “Extended controls” («Расширенные элементы управления»), обозначенную тремя точками, выберите “Phone” («Телефон»), а затем “Call” («Вызов»). Номер телефона можно изменить.



### 44.3. Открытие диспетчера AVD

После установки SDK можно открыть AVD Manager из командной строки с помощью команды `android avd`.

Доступ к AVD Manager можно также получить из Android studio, используя Tools > Android > AVD Manager или нажав на значок AVD Manager на панели инструментов, который является вторым на скриншоте ниже.



### 44.4. Устранение ошибок при запуске эмулятора

Прежде всего убедитесь, что в настройках BIOS включена функция “Виртуализация”.

Запустите **Android SDK Manager**, выберите **Extras** и затем выберите **Intel Hardware Accelerated Execution Manager**, дождитесь завершения загрузки. Если все еще не работает, откройте папку **SDK** и выполните `/extras/intel/Hardware_Accelerated_Execution_Manager/IntelHAXM.exe`.

Для завершения установки следуйте инструкциям на экране.

Для OS X это можно сделать без экранных подсказок, например так:

```
/extras/intel/Hardware_Accelerated_Execution_Manager/HAXM\ installation
```

Если ваш процессор не поддерживает VT-x или SVM, вы не можете использовать образы Android на базе архитектуры x86. Вместо этого используйте образы на базе ARM.

После завершения установки убедитесь в корректной работе драйвера виртуализации, открыв окно командной строки и выполнив следующую команду: `sc query intelhaxm`.

Для запуска эмулятора на базе x86 с VM-ускорением: если вы запускаете эмулятор из командной строки, просто укажите AVD на базе x86: `emulator -avd <avd_name>`.

Если вы правильно выполнили все описанные выше действия, то, несомненно, должны увидеть, что ваш AVD с HAXM работает должным образом.

## Глава 45. Службы

Службы работают в фоновом режиме для выполнения длительных операций или удаленных процессов. Они не предоставляют пользовательского интерфейса, а производят активность только в фоновом режиме с участием данных пользователя. Например, служба может проигрывать музыку в фоновом режиме, пока пользователь находится в другом приложении, или загружать данные из Интернета, не блокируя взаимодействие пользователя с Android-устройством.

### 45.1. Жизненный цикл службы

Жизненный цикл служб включает в себя следующие обратные вызовы:

- `onCreate()`:

Выполняется при первом создании службы для установки необходимых начальных конфигураций. Этот метод выполняется только в том случае, если служба еще не запущена.

- `onStartCommand()`:

Выполняется каждый раз, когда `startService()` вызывается другим компонентом, например активностью или широковещательным приемником. При использовании этого метода служба будет работать до тех пор, пока вы не вызовете `stopSelf()` или `stopService()`. Обра-

тите внимание, что независимо от количества вызовов `onStartCommand()` методы `stopSelf()` и `stopService()` должны быть вызваны только один раз для остановки службы.

- `onBind()`:

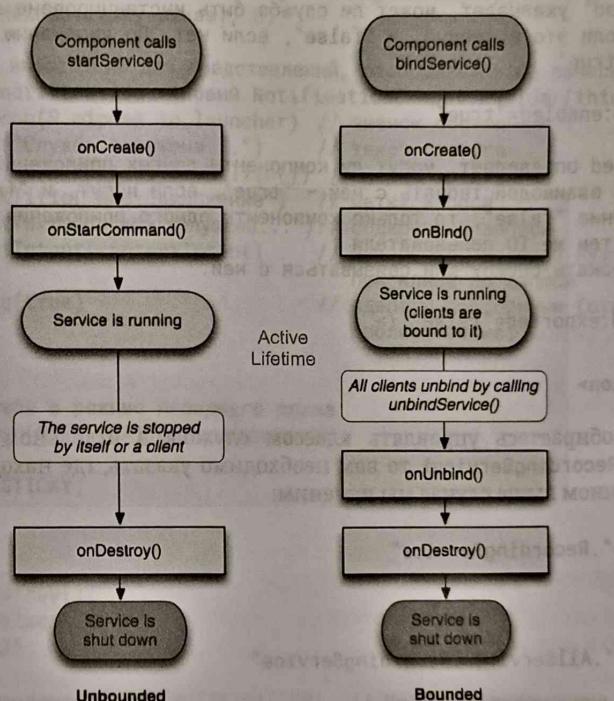
Выполняется при вызове компонентом функции `bindService()` и возвращает экземпляр `IBinder`, обеспечивая канал связи со службой. Вызов `bindService()` будет поддерживать работу службы до тех пор, пока существуют привязанные к нему клиенты.

- `onDestroy()`:

Выполняется, когда служба больше не используется, и позволяет избавиться от выделенных ресурсов.

Важно отметить, что в течение жизненного цикла службы могут быть вызваны и другие обратные вызовы, такие как `onConfigurationChanged()` и `onLowMemory()`.

Подробнее на эту тему см. <https://developer.android.com/guide/components/services.html>.



## 45.2. Определение процесса службы

Поле `android:process` определяет имя процесса, в котором будет запущена служба. Обычно все компоненты приложения запускаются в процессе по умолчанию, созданном для данного приложения. Однако компонент может переопределить значение по умолчанию с помощью собственного атрибута `process`, что позволяет распределить приложение по нескольким процессам.

Если имя, присвоенное атрибуту, начинается с двоеточия (‘:’), то служба будет выполняться в отдельном процессе.

```
<service
    android:name="com.example.appName"
    android:process=":externalProcess" />
```

Если имя процесса начинается со строчного символа, то служба будет запущена в глобальном процессе с таким именем, если у нее есть на это разрешение. Это позволяет компо-

нентам различных приложений совместно использовать один процесс, что снижает потребление ресурсов.

### 45.3. Создание несвязанной службы

Первое, что необходимо сделать, – добавить службу в `AndroidManifest.xml`, внутри тега `<application>`:

```

<application ...>
    ...
    <service
        android:name=".RecordingService"
        // Тег "enabled" указывает, может ли служба быть инстанцирована системой.
        // "true" - если это возможно, и "false", если нет. По умолчанию используется
        // значение "true"

        android:enabled="true"

        // тег exported определяет, могут ли компоненты других приложений вызывать
        // службу или взаимодействовать с ней - "true", если могут, и "false", если нет.
        // если значение "false", то только компоненты одного приложения или приложения
        // с одним и тем же ID пользователя
        // могут запускать службу или связываться с ней.

        android:exported="false" />

    </application>

```

Если вы собираетесь управлять классом службы в отдельном пакете (например, `.AllServices.RecordingService`), то вам необходимо указать, где находится ваша служба. Так, в приведенном выше случае мы изменим:

```
    android:name=".RecordingService"
```

на:

```
    android:name=".AllServices.RecordingService"
```

...или самый простой способ – указать полное имя пакета. Затем мы создаем собственно класс службы:

```

public class RecordingService extends Service {
    private int NOTIFICATION=1;// Уникальный идентификатор для нашего уведомления

    public static boolean isRunning = false;
    public static RecordingService instance = null;

    private NotificationManager notificationManager = null;

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public void onCreate(){

```

```

instance = this;
isRunning = true;

notificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

super.onCreate();
}

@Override
public int onStartCommand(Intent intent, int flags, int startId){
    // PendingIntent запускает нашу активность, если пользователь выберет
    // это уведомление
    PendingIntent contentIntent = PendingIntent.getActivity(this, 0, new
    Intent(this, MainActivity.class), 0);

    // Установите информацию для представлений, отображаемых на панели уведомлений
    Notification notification = новый NotificationCompat.Builder(this)
        .setSmallIcon(R.mipmap.ic_launcher) // значок состояния
        .setTicker("Служба запущена...") // текст статуса
        .setWhen(System.currentTimeMillis()) // метка времени
        .setContentTitle("Мое приложение") // метка записи
        .setContentText("Служба запущена...")// содержание записи
        .setContentIntent(contentIntent) // намерение, которое нужно отправить
                                         // при клике по записи
        .setOngoing(true) // сделать постоянным (отключить
                           // пролистывание)
        .build();

    // Запуск службы в режиме переднего плана
    startForeground(NOTIFICATION, уведомление);

    return START_STICKY;
}

@Override
public void onDestroy(){
    isRunning = false;
    instance = null;

    notificationManager.cancel(NOTIFICATION); // Удалить уведомление
    super.onDestroy();
}

public void doSomething(){
    Toast.makeText(getApplicationContext(), "Выполнение задания из службы...", Toast.LENGTH_SHORT).show();
}
}

```

Все, что делает эта служба, – показывает уведомление, когда она запущена, и может выводить тосты при вызове ее метода `doSomething()`. Как можно заметить, она реализована как синглтон, отслеживающий свой собственный экземпляр – но без обычного метода статической фабрики синглтонов, поскольку службы являются естественными синглтонами и создаются намерениями. Внешний экземпляр нужен для того, чтобы получить “дескриптор” к службе, когда она запущена.

Наконец, нам необходимо запустить и остановить службу из активности:

```
public void startOrStopService(){
    if(RecordingService.isRunning){
        // Остановить службу
        Intent intent = new Intent(this, RecordingService.class);
        stopService(intent);
    }
    else {
        // Запустить службу
        Intent intent = new Intent(this, RecordingService.class);
        startService(intent);
    }
}
```

В данном примере служба запускается и останавливается одним и тем же методом в зависимости от его текущего состояния.

Можно также вызывать метод `doSomething()` из нашей активности:

```
public void makeServiceDoSomething(){
    if( RecordingService.isRunning )
        RecordingService.instance.doSomething();
}
```

## 45.4. Запуск службы

Запустить службу очень просто – достаточно вызвать `startService` с намерением, находясь внутри активности:

```
Intent intent = new Intent(this, MyService.class); // замените MyService на имя
// вашей службы
intent.putExtra(Intent.EXTRA_TEXT, "Some text"); // добавить любые дополнительные
// данные для передачи в службу

startService(intent); // Вызовите startService для запуска службы
```

## 45.5. Создание связанной службы с помощью Binder

Создайте класс, расширяющий класс `Service`, и в переопределенном методе `onBind` верните локальный экземпляр `Binder`:

```
public class LocalService extends Service {
    // Binder, передаваемый клиентам
    private final IBinder mBinder = new LocalBinder();

    /**
     * Класс, используемый для клиента Binder. Поскольку мы знаем, что эта служба
     * всегда работает в том же процессе, что и его клиенты, нам не нужно иметь
     * дело с IPC.
     */
    public class LocalBinder extends Binder {
        LocalService getService() {
            // Возвращаем данный экземпляр LocalService, чтобы клиенты могли вызывать
            // открытые методы
            return LocalService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
```

```

        return mBinder;
    }
}

Затем в своей активности привязываемся к службе в обратном вызове onStart, используя экземпляр ServiceConnection, и отвязываемся от нее в onStop:

public class BindingActivity extends Activity {
    LocalService mService;
    boolean mBound = false;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    protected void onStart() {
        super.onStart();
        // Привязка к LocalService
        Intent intent = new Intent(this, LocalService.class);
        bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
        super.onStop();
        // Отвязка от службы
        if (mBound) {
            unbindService(mConnection);
            mBound = false;
        }
    }

    /**
     * Определяет обратные вызовы для привязки службы, передаваемые в bindService()
     */
    private ServiceConnection mConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName className, IBinder service) {
            // Мы привязались к LocalService, приводим IBinder и получаем экземпляр
            LocalService
            LocalBinder binder = (LocalBinder) service;
            mService = binder.getService();
            mBound = true;
        }

        @Override
        public void onServiceDisconnected(ComponentName arg0) {
            mBound = false;
        }
    };
}

```

## 45.6. Создание удаленной службы (через AIDL)

Опишем интерфейс доступа к службе с помощью файла .aidl:

```
// IRemoteService.aidl
package com.example.android;
```

```
// Объявим здесь все типы, не являющиеся типами по умолчанию, с помощью операторов
импорта

/** Пример интерфейса службы */
interface IRemoteService {
    /** Запросить идентификатор процесса данной службы. */
    int getPid();
}
```

Теперь после сборки приложения инструменты sdk генерируют соответствующий java-файл. Этот файл будет содержать класс Stub, который реализует наш интерфейс aidl и который нам необходимо расширить:

```
public class RemoteService extends Service {

    private final IRemoteService.Stub binder = new IRemoteService.Stub() {
        @Override
        public int getPid() throws RemoteException {
            return Process.myPid();
        }
    };
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }
}
```

Затем в активности:

```
public class MainActivity extends AppCompatActivity {
    private final ServiceConnection connection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName componentName, IBinder iBinder) {
            IRemoteService service = IRemoteService.Stub.asInterface(iBinder);
            Toast.makeText(this, "Activity process: " + Process.myPid + ", Service
process: " + getRemotePid(service), LENGTH_SHORT).show();
        }

        @Override
        public void onServiceDisconnected(ComponentName componentName) {}
    };

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

    @Override
    protected void onStart() {
        super.onStart();
        Intent intent = new Intent(this, RemoteService.class);
        bindService(intent, connection, Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
```

```

super.onStop();
unbindService(connection);
}

private int getRemotePid(IRemoteService service) {
    int result = -1;

    try {
        result = service.getPid();
    } catch (RemoteException e) {
        e.printStackTrace();
    }

    return result;
}
}

```

## Глава 46. Файл манифеста

Манифест (manifest) – это обязательный файл, который называется `AndroidManifest.xml` и располагается в корневом каталоге приложения. В нем указываются имя приложения, икона, имя Java-пакета, версия, декларация активностей, служб, разрешений приложения и другая информация.

### 46.1. Объявление компонентов

Основная задача манифеста – информировать систему о компонентах приложения. Например, в файле манифеста активность может быть объявлена следующим образом:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
            android:label="@string/example_label" ... >
        </activity>
        ...
    </application>
</manifest>

```

В элементе `<application>` атрибут `android:icon` указывает на ресурсы для иконки, идентифицирующей приложение. Атрибут `android:name` задает полное имя класса подкласса активности, а атрибут `android:label` – строку, используемую в качестве видимой пользователем метки для активности.

Все компоненты приложения необходимо объявлять следующим образом:

- `<activity>` – элементы для активностей;
- `<service>` – элементы для служб;
- `<receiver>` – элементы для широковещательных приемников;
- `<provider>` – элементы для поставщиков контента.

Действия, службы и поставщики контента, включенные в исходный код, но не объявленные в манифесте, не видны системе и, следовательно, не могут быть запущены. Однако широковещательные приемники могут быть либо объявлены в манифесте, либо созданы динамически в коде (как объекты `BroadcastReceiver`) и зарегистрированы в системе вызовом `registerReceiver()`.

## 46.2. Объявление разрешений в файле манифеста

Любое разрешение, необходимое вашему приложению для доступа к защищенной части API или для взаимодействия с другими приложениями, должно быть объявлено в файле `AndroidManifest.xml`. Для этого используется тег `<uses-permission />`.

### Синтаксис

```
<uses-permission android:name="string"
    android:maxSdkVersion="integer"/>
```

`android:name`: имя требуемого разрешения.

`android:maxSdkVersion`: наивысший уровень API, на котором данное разрешение должно быть предоставлено вашему приложению. Установка этого разрешения не является обязательной и должна выполняться только в том случае, если разрешение, необходимое вашему приложению, больше не требуется на определенном уровне API.

Образец файла `AndroidManifest.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.samplepackage">

    <!-- запрос разрешения на использование Интернета -->
    <uses-permission android:name="android.permission.INTERNET" />

    <!-- запрос разрешения на использование камеры -->
    <uses-permission android:name="android.permission.CAMERA" />

    <!-- запрос разрешения на запись во внешнее хранилище -->
    <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE"
        android:maxSdkVersion="18" />

    <application>....</application>
</manifest>
```

# Глава 47. Gradle для Android

Gradle – это система сборки на базе Java Virtual Machine, позволяющая разработчикам писать высокоуровневые скрипты, которые можно использовать для автоматизации процесса компиляции и создания приложений. Это гибкая система, основанная на плагинах, которая позволяет автоматизировать различные аспекты процесса сборки, включая компиляцию и подписание .jar, загрузку и управление внешними зависимостями, введение полей в `AndroidManifest` или использование определенных версий SDK.

## 47.1. Базовый файл `build.gradle`

Приведем пример стандартного файла `build.gradle` в модуле.

```
apply plugin: 'com.android.application'

android {
```

```

compileSdkVersion 25
buildToolsVersion '25.0.3'

signingConfigs {
    applicationName {
        keyAlias 'applicationName'
        keyPassword 'password'
        storeFile file('../key/applicationName.jks')
        storePassword 'keystorePassword'
    }
}
defaultConfig {
    applicationId 'com.company.applicationName'
    minSdkVersion 14
    targetSdkVersion 25
    versionCode 1
    versionName '1.0'
    signingConfig signingConfigs.applicationName
}
buildTypes {
    release {
        minifyEnabled true
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
        rules.pro'
    }
}
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])

    compile 'com.android.support:appcompat-v7:25.3.1'
    compile 'com.android.support:design:25.3.1'

    testCompile 'junit:junit:4.12'
}

```

### DSL (domain-specific language)

Каждый блок в приведенном выше файле называется DSL (domain-specific language, предметно-ориентированный язык).

#### Плагины

Первая строка, `apply plugin: 'com.android.application'`, применяет плагин Android для Gradle к сборке и делает блок `android {}` доступным для объявления опций сборки, специфичных для Android.

#### Для приложения Android:

```
apply plugin: 'com.android.application'
```

#### Для библиотеки Android:

```
apply plugin: 'com.android.library'
```

#### Понимание DSL в приведенном выше примере

Вторая часть, блок `android {...}`, представляет собой Android DSL, содержащий информацию о вашем проекте.

Например, в нем можно задать параметр `compileSdkVersion`, определяющий уровень API Android, который должен использоваться Gradle для компиляции приложения.

В подблоке `defaultConfig` хранятся значения по умолчанию для вашего манифеста. Вы можете переопределить (`override`) их с помощью вариантов (Product Flavors).

## Зависимости

Блок зависимостей `dependencies` определен вне блока `android { ... }`: это означает, что он определен не плагином Android, а стандартным Gradle.

В блоке зависимостей указывается, какие внешние библиотеки (как правило, библиотеки Android, но можно использовать и библиотеки Java) вы хотите включить в свое приложение. Gradle автоматически загрузит эти зависимости за вас (если нет локальной копии), вам нужно будет просто добавить соответствующие строки `compile`, если вы захотите добавить другую библиотеку.

Рассмотрим одну из таких строк:

```
compile 'com.android.support:design:25.3.1'
```

В этой строке, по сути, говорится:

добавить в мой проект зависимость от библиотеки Android support design library.

Gradle обеспечит загрузку и присутствие библиотеки, чтобы вы могли использовать ее в своем приложении, и ее код также будет включен в ваше приложение.

Если вы знакомы с Maven, то этот синтаксис выглядит так: *GroupId*, двоеточие, *ArtifactId*, еще одно двоеточие, затем версия зависимости, которую вы хотите включить, что дает вам полный контроль над версионированием.

Хотя можно указывать версии артефактов с помощью знака плюс (+), лучше этого не делать; это может привести к проблемам, если библиотека будет обновлена с разрушающими изменениями без вашего ведома, что, скорее всего, приведет к сбоям в работе вашего приложения.

Вы можете добавлять различные виды зависимостей:

- локальные бинарные зависимости
- модульные зависимости
- удаленные зависимости

Особое внимание следует уделить зависимостям `aar` `flat`.

Примечание по поводу параметра `-v7` в `appcompat-v7`.

```
compile 'com.android.support:appcompat-v7:25.3.1'
```

Это просто означает, что данная библиотека (`appcompat`) совместима с Android API уровня 7 и выше.

Замечание по поводу `junit:junit:4.12`

Это зависимость для модульного тестирования.

## Указание зависимостей, характерных для различных конфигураций сборки

Вы можете указать, что зависимость должна использоваться только для определенной конфигурации сборки, или определить различные зависимости для типов сборки или разновидностей продукта (например, `debug`, `test` или `release`), используя `debugCompile`, `testCompile` или `releaseCompile` вместо обычного `compile`.

Это позволяет не включать в сборку тестовые и отладочные зависимости, что делает APK максимально компактным и гарантирует, что отладочная информация не может быть использована для получения внутренней информации о вашем приложении.

## `signingConfig`

`SigningConfig` позволяет настроить Gradle на включение информации о хранилище ключей `keystore` и гарантировать, что APK, созданные с использованием этих конфигураций, подписаны и готовы к выпуску в Play Store.

**Замечание:** не рекомендуется хранить конфигурации подписания внутри файла Gradle. Чтобы удалить конфигурации подписания, просто опустите часть `signingConfigs`.

Их можно задавать различными способами:

- хранить во внешнем файле;
- сохранять в переменных окружения.

## 47.2. Определение и использование полей конфигурации сборки

### BuildConfigField

Gradle позволяет в строках `buildConfigField` определять константы. Эти константы будут доступны во время выполнения как статические поля класса `BuildConfig`. Это может быть использовано для создания вариантов (*Flavors*), при помощи определения всех полей в блоке `defaultConfig`, а затем переопределением их для отдельных вариантов сборки по мере необходимости.

В этом примере определяется дата сборки и устанавливается флаг сборки для производства, а не для тестирования:

```
android {
    ...
    defaultConfig {
        ...
        // определение даты сборки
        buildConfigField "long", "BUILD_DATE", System.currentTimeMillis() + "L"
        // определить, является ли данная сборка производственной
        buildConfigField "boolean", "IS_PRODUCTION", "false"
        // обратите внимание, что для определения строки необходимо ее экранировать
        buildConfigField "String", "API_KEY", "\"my_api_key\""
    }

    productFlavors {
        prod {
            // переопределяем производственный флаг для варианта "prod"
            buildConfigField "boolean", "IS_PRODUCTION", "true"
            resValue 'string', 'app_name', 'My App Name'
        }
        dev {
            // наследование полей по умолчанию
            resValue 'string', 'app_name', 'My App Name - Dev'
        }
    }
}
```

Автоматически сгенерированный файл `<package_name>.BuildConfig.java` в папке `gen` содержит следующие поля, основанные на приведенной выше директиве:

```
public class BuildConfig {
    // ... другие генерируемые поля ...
    public static final long BUILD_DATE = 1469504547000L;
    public static final boolean IS_PRODUCTION = false;
    public static final String API_KEY = "my_api_key";
}
```

Теперь заданные поля можно использовать в приложении во время выполнения, обратившись к сгенерированному классу `BuildConfig`:

```
public void example() {
    // форматирование даты сборки
```

```

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd");
String buildDate = dateFormat.format(new Date(BuildConfig.BUILD_DATE));
Log.d("build date", buildDate);

// сделать что-то в зависимости от того, является ли это производственной сборкой
if (BuildConfig.IS_PRODUCTION) {
    connectToProductionApiEndpoint();
} else {
    connectToStagingApiEndpoint();
}
}

```

### ResValue

Параметр `resValue` в вариантах `productFlavors` создает значение ресурса. Это может быть ресурс любого типа(`string`, `dimen`, `color` и т. д.). Это аналогично определению ресурса в соответствующем файле: например, определение строки `string` в файле `strings.xml`. Преимущество в том, что ресурс, определенный в `gradle`, может быть изменен в зависимости от вашего `productFlavor/buildVariant`. Чтобы получить доступ к значению, напишите тот же код, что и для доступа к `res` из файла `resources`:

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

Важно, что ресурсы, определенные таким образом, не могут модифицировать существующие ресурсы, определенные в файлах. Они могут только создавать новые значения ресурсов.

Для некоторых библиотек (например, Google Maps Android API) требуется ключ API, предоставляемый в манифесте в виде тега метаданных `meta-data`. Если для отладочной и производственной сборки требуются разные ключи, укажите заполняемый Gradle заполнитель манифеста.

В файле `AndroidManifest.xml`:

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="${MAPS_API_KEY}" />
```

...а затем настройте соответствующее поле в файле `build.gradle`:

```
android {
    defaultConfig {
        ...
        // Ваш ключ разработки
        manifestPlaceholders = [ MAPS_API_KEY: "AIza..." ]
    }

    productFlavors {
        prod {
            // Ваш производственный ключ
            manifestPlaceholders = [ MAPS_API_KEY: "AIza..." ]
        }
    }
}
```

Система сборки Android автоматически генерирует ряд полей и помещает их в файл `BuildConfig.java`.

Этими полями являются:

Поле	Описание
DEBUG	булево значение, указывающее, находится ли приложение в режиме отладки или в режиме выпуска
APPLICATION_ID	строка с идентификатором приложения (например <code>com.example.app</code> )

Поле	Описание
BUILD_TYPE	строка с типом сборки приложения (обычно либо debug, либо release)
FLAVOR	строка, содержащая конкретный вариант (flavor) сборки
VERSION_CODE	целочисленное значение, содержащее номер версии (сборки). Это то же самое, что versionCode в build.gradle или versionCode в AndroidManifest.xml
VERSION_NAME	строка, содержащая имя версии (сборки). Это то же самое, что и versionName в build.gradle или versionName в AndroidManifest.xml

В дополнение к вышесказанному, если вы определили несколько измерений варианта, то каждое из них будет иметь свое собственное значение. Например, если у вас есть два измерения варианта – цвет и размер, то у вас также будут следующие переменные:

Поле	Описание
FLAVOR_color	строка, содержащая значение для варианта 'color'.
FLAVOR_size	строка, содержащая значение для варианта 'size'.

## 47.3. Централизация зависимостей с помощью файла dependencies.gradle

При работе с многомодульными проектами полезно централизовать зависимости в одном месте, а не распределять их по многим файлам сборки, особенно для таких распространенных библиотек, как библиотеки поддержки Android и библиотеки Firebase.

Одним из рекомендуемых способов является разделение файлов сборки Gradle: один build.gradle для каждого модуля, а также один в корне проекта и еще один, например, для зависимостей:

```
root
    |   project loadFile('file:///path/to/file.gradle')
    +- gradleScript/
        |       dependencies.gradle
    +- module1/
        |       build.gradle
    +- модуль2/
        |       build.gradle
    +- build.gradle
```

Затем все ваши зависимости могут быть расположены в gradleScript/dependencies.gradle:

```
ext {
    // Версия
    supportVersion = '24.1.0'

    // Поддержка зависимостей библиотек
    supportDependencies = [
        design: "com.android.support:design:${supportVersion}",
        recyclerView: "com.android.support:recyclerview-v7:${supportVersion}",
        cardView: "com.android.support:cardview-v7:${supportVersion}",
        appCompat: "com.android.support:appcompat-v7:${supportVersion}",
        supportAnnotation: "com.android.support:support-annotations:${supportVersion}"
    ]

    firebaseVersion = '9.2.0';

    firebaseDependencies = [
        core: "com.google.firebaseio:firebase-core:${firebaseVersion}",
        database: "com.google.firebaseio:firebase-database:${firebaseVersion}"
    ]
}
```

```

storage:      "com.google.firebaseio.firebaseio-storage:${firebaseVersion}",
crash:       "com.google.firebaseio.firebaseio-crash:${firebaseVersion}",
auth:        "com.google.firebaseio.firebaseio-auth:${firebaseVersion}",
messaging:   "com.google.firebaseio.firebaseio-messaging:${firebaseVersion}",
remoteConfig: "com.google.firebaseio.firebaseio-config:${firebaseVersion}",
invites:    "com.google.firebaseio.firebaseio-invites:${firebaseVersion}",
adMod:      com.google.firebaseio.firebaseio-ads:${firebaseVersion}",
appIndexing: "com.google.android.gms.play-services-appindexing:${firebase
Version}",

];
}

```

...который затем может быть применен из этого файла в файле верхнего уровня build.gradle следующим образом:

```
// Загрузка зависимостей
apply from: 'gradleScript/dependencies.gradle'
```

...и в модуле module1/build.gradle следующим образом:

```
// Файл сборки модуля
dependencies {
    // ...
    compile supportDependencies.appcompat
    compile supportDependencies.design
    compile firebaseDependencies.crash
}
```

### Другой подход

Менее «многословный» подход к централизации версий библиотечных зависимостей может быть достигнут путем однократного объявления номера версии в качестве переменной и использования его повсеместно.

В корне рабочего пространства build.gradle добавьте следующее:

```
ext.v = [
    supportVersion:'24.1.1',
]
```

...и в каждый модуль, использующий одну и ту же библиотеку, добавьте необходимые библиотеки:

```
compile "com.android.support:support-v4:${v.supportVersion}"
compile "com.android.support:recyclerview-v7:${v.supportVersion}"
compile "com.android.support:design:${v.supportVersion}"
compile "com.android.support:support-annotations:${v.supportVersion}"
```

## 47.4. Подписание APK без раскрытия пароля хранилища ключей

Конфигурацию подписи apk можно задать в файле build.gradle с помощью этих свойств:

- storeFile: файл хранилища ключей
- storePassword: пароль хранилища ключей
- keyAlias: имя псевдонима ключа
- keyPassword: пароль псевдонима ключа

Во многих случаях может потребоваться избегать подобной информации в файле build.gradle.

**Метод А: Настройка подписи релизов с помощью файла keystore.properties**

Можно настроить build.gradle таким образом, чтобы приложение считывало информацию о конфигурации подписи из файла свойств, например keystore.properties.

Подобная установка подписи выгодна тем, что:

- информация о конфигурации подписи находится отдельно от файла build.gradle;
- вам не нужно вмешиваться в процесс подписания, чтобы предоставить пароли для файла keystore;
- вы можете легко исключить файл keystore.properties из контроля версий.

Сначала создайте в корне проекта файл keystore.properties с содержанием, подобным этому (замените значения на свои):

```
storeFile=keystore.jks
storePassword=storePassword
keyAlias=keyAlias
keyPassword=keyPassword
```

Теперь в файле build.gradle вашего приложения настройте блок signingConfigs следующим образом:

```
android {
    ...
    signingConfigs {
        release {
            def propsFile = rootProject.file('keystore.properties')
            if (propsFile.exists()) {
                def props = new Properties()
                props.load(new FileInputStream(propsFile))
                storeFile = file(props['storeFile'])
                storePassword = props['storePassword']
                keyAlias = props['keyAlias']
                keyPassword = props['keyPassword']
            }
        }
    }
}
```

Вот, собственно, и все, но не забудьте исключить как файл keystore, так и файл keystore.properties из системы контроля версий.

Следует отметить несколько моментов.

- Путь storeFile, указанный в файле keystore.properties, должен быть относительным к файлу build.gradle вашего приложения. В данном примере предполагается, что файл keystore находится в том же каталоге, что и файл build.gradle приложения.
- В данном примере файл keystore.properties находится в корне проекта. Если вы разместите его в другом месте, то обязательно измените значение в rootProject.file('keystore.properties') на свое, относительно корня проекта.

**Метод В: Использование переменной среды**

То же самое можно сделать и без файла свойств, что затрудняет подбор пароля:

```
android {
    ...
    signingConfigs {
        release {
```

```

        storeFile file('/your/keystore/location/key')
        keyAlias 'your_alias'
        String ps = System.getenv("ps")
        if (ps == null) {
            throw new GradleException('missing ps env variable')
        }
        keyPassword ps
        storePassword ps
    }
}

```

Переменная окружения "ps" может быть глобальной, но более безопасным подходом может быть добавление ее только в оболочку Android Studio.

В Linux это можно сделать, отредактировав вход на рабочий стол (Desktop Entry) Android Studio:

```
Exec=sh -c "export ps=myPassword123 ; /path/to/studio.sh"
```

## 47.5. Добавление зависимостей, специфичных для варианта продукта

Зависимости могут быть добавлены для конкретного продукта, аналогично тому, как они могут быть добавлены для конкретных конфигураций сборки.

Для данного примера предположим, что мы уже определили два варианта (flavors) продукта – бесплатный и платный. Затем мы можем добавить зависимость AdMob для бесплатного варианта и библиотеку Picasso для платного следующим образом:

```

android {
    ...
    productFlavors {
        free {
            applicationId "com.example.app.free"
            versionName "1.0-free"
        }
        paid {
            applicationId "com.example.app.paid"
            versionName "1.0-paid"
        }
    }
    ...
    dependencies {
        ...
        // Добавить AdMob только для бесплатного
        freeCompile 'com.android.support:appcompat-v7:23.1.1'
        freeCompile 'com.google.android.gms:play-services-ads:8.4.0'
        freeCompile 'com.android.support:support-v4:23.1.1'
        ...
        // Добавить picasso только для платного
        paidCompile 'com.squareup.picasso:picasso:2.5.2'
    }
    ...
}

```

## 47.6. Указание различных идентификаторов приложений для типов сборки и вариантов продукта

Вы можете указать различные идентификаторы приложений или имена пакетов для каждого `buildType` или `productFlavor`, используя конфигурационный атрибут `applicationIdSuffix`.

Пример суффиксации `applicationId` для каждого `buildType`:

```
defaultConfig {
    applicationId "com.package.android"
    minSdkVersion 17
    targetSdkVersion 23
    versionCode 1
    versionName "1.0"
}

buildTypes {
    release {
        debuggable false
    }
    development {
        debuggable true
        applicationIdSuffix ".dev"
    }
    testing {
        debuggable true
        applicationIdSuffix ".qa"
    }
}
```

Теперь наши результирующие `applicationIds` будут такими:

- `com.package.android` для `release` (выпуска)
- `com.package.android.dev` для `development` (разработки)
- `com.package.android.qa` для `testing` (тестирования)

Это можно сделать и для `productFlavors` (вариантов):

```
productFlavors {
    free {
        applicationIdSuffix ".free"
    }
    paid {
        applicationIdSuffix ".paid"
    }
}
```

В результате будут получены следующие идентификаторы приложений (`applicationIds`):

- `com.package.android.free` для бесплатного варианта
- `com.package.android.paid` для платного

## 47.7. Версионирование сборок с помощью файла `version.properties`

С помощью Gradle можно автоматически увеличивать версию пакета при каждой сборке. Для этого создайте файл `version.properties` в том же каталоге, что и `build.gradle`, со следующим содержимым:

```
VERSION_MAJOR=0
VERSION_MINOR=1
VERSION_BUILD=1
```

(Изменяем значения MAJOR и MINOR по своему усмотрению). Затем в файле build.gradle добавим следующий код в раздел android:

```
// Считывание информации о версии из локального файла и ее увеличение по мере
// необходимости
def versionPropsFile = file('version.properties')
if (versionPropsFile.canRead()) {
    def Properties versionProps = new Properties()
    versionProps.load(new FileInputStream(versionPropsFile))

    def versionMajor = versionProps['VERSION_MAJOR'].toInteger()
    def versionMinor = versionProps['VERSION_MINOR'].toInteger()
    def versionBuild = versionProps['VERSION_BUILD'].toInteger() + 1

    // Обновление номера сборки в локальном файле
    versionProps['VERSION_BUILD'] = versionBuild.toString()
    versionProps.store(versionPropsFile.newWriter(), null)

    defaultConfig { versionCode versionBuild
        versionName "${versionMajor}.${versionMinor}." + String.format("%05d",
            versionBuild)
    }
}
```

Эта информация может быть доступна в Java в виде строки BuildConfig.VERSION\_NAME для полного {major}.{minor}.{build} номера и как целое число BuildConfig.VERSION\_CODE только для номера сборки.

## 47.8. Определение вариантов продукта

Варианты продукта определяются в файле build.gradle внутри блока android { ... }, как показано ниже:

```
...
android {
    ...
    productFlavors {
        free {
            applicationId "com.example.app.free"
            versionName "1.0-free"
        }
        paid {
            applicationId "com.example.app.paid"
            versionName "1.0-paid"
        }
    }
}
```

Таким образом, у нас появилось два дополнительных вида (варианта) продукта: бесплатный (free) и платный (paid). Каждый из них может иметь свою специфическую конфигурацию и атрибуты. Например, у обоих наших новых вариантов есть отдельные applicationId и versionName, отличающиеся от таковых у существующего основного варианта (доступного по умолчанию, поэтому здесь не показанного).

## 47.9. Изменение выходного имени apk и добавление имени версии

Приведем код для изменения имени выходного файла приложения (.apk). Имя может быть сконфигурировано путем присвоения другому значению параметра newName:

```
android {
    applicationVariants.all { variant ->
        def newName = "ApkName";
        variant.outputs.each { output ->
            def apk = output.outputFile;

            newName += "-v" + defaultConfig.versionName;
            if (variant.buildType.name == "release") {
                newName += "-release.apk";
            } else {
                newName += ".apk";
            }
            if (!output.zipAlign) {
                newName = newName.replace(".apk", "-unaligned.apk");
            }

            output.outputFile = new File(apk.parentFile, newName);
            logger.info("INFO: Set outputFile to "
                    + output.outputFile
                    + " для [" + output.name + "]");
        }
    }
}
```

## 47.10. Добавление ресурсов, специфичных для варианта продукта

Ресурсы могут быть добавлены для определенного варианта программного продукта.

Для данного примера предположим, что мы уже определили два варианта продукта – бесплатный и платный. Для добавления ресурсов, специфичных для каждого конкретного варианта, создаем дополнительные папки ресурсов наряду с основной папкой main/res, в которые затем можем добавлять ресурсы, как обычно. В данном примере мы определим строку status для каждого варианта продукта:

/src/main/res/values/strings.xml

```
<resources>
    <string name="status">Default</string>
</resources>
```

/src/free/res/values/strings.xml

```
<resources>
    <string name="status">Free</string>
</resources>
```

/src/paid/res/values/strings.xml

```
<resources>
    <string name="status">Paid</string>
</resources>
```

Строки статуса (`status`), специфичные для конкретного продукта, будут переопределять значение статуса в основном (`main`) варианте.

## 47.11. Почему в проекте Android Studio два файла `build.gradle`?

`<PROJECT_ROOT>\app\build.gradle` специфичен для модуля приложения.

`<PROJECT_ROOT>\build.gradle` – это “файл сборки верхнего уровня”, в который можно добавить параметры конфигурации, общие для всех подпроектов/модулей.

Если вы используете в проекте другой модуль в качестве локальной библиотеки, то у вас будет другой файл `build.gradle`:

`<PROJECT_ROOT>\module\build.gradle`

В файле верхнего уровня можно указать общие свойства как блока `buildscript`, так и некоторые другие общие свойства:

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.2.0'
        classpath 'com.google.gms:google-services:3.0.0'
    }
}

ext {
    compileSdkVersion = 23
    buildToolsVersion = "23.0.1"
}
```

В `app\build.gradle` задаются только свойства модуля:

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion rootProject.ext.compileSdkVersion
    buildToolsVersion rootProject.ext.buildToolsVersion
}

dependencies {
    //....
}
```

## 47.12. Структура каталогов для ресурсов, специфичных для варианта продукта

Различные варианты сборки приложений могут содержать различные ресурсы. Чтобы создать ресурс для конкретного варианта, создайте в каталоге `src` каталог с именем вашего варианта в нижнем регистре и добавьте в него свои ресурсы так же, как это делается обычно.

Например, если у вас есть вариант `Development` и вы хотите предоставить для него отдельную иконку запуска, то создайте каталог `src/development/res/drawable-mdpi` и внутри этого каталога создайте файл `ic_launcher.png` с иконкой, специфичной для вашего `Development`-варианта.

Структура каталогов будет выглядеть следующим образом:

```
src/
  main/
```

```

res/
    drawable-mdpi/
        ic_launcher.png <-- иконка запуска по умолчанию
development/
    res/
        drawable-mdpi/
            ic_launcher.png <-- иконка запуска, используемая в случае, если продукт
имеет вариант Development

```

Разумеется, в этом случае необходимо будет также создать иконки для `drawable-hdpi`, `drawable-xhdpi` и т. д.

## 47.13. Включение Proguard с помощью gradle

Для включения конфигурации Proguard в приложении необходимо включить ее в `gradle`-файле на уровне модуля. Для этого необходимо установить значение `minifyEnabled` в `true`.

```

buildTypes {
    release {
        minifyEnabled true
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
        rules.pro'
    }
}

```

Приведенный выше код применит к apk конфигурации Proguard, содержащиеся в стандартном Android SDK, в сочетании с файлом `proguard-rules.pro` вашего модуля.

## 47.14. Игнорирование варианта сборки

По некоторым причинам вы можете игнорировать варианты сборки. Например, у вас есть 'mock', «макетный» вариант продукта, и вы используете его только для отладки, например, для модульных/инструментальных тестов.

Игнорируем вариант `mockRelease` из нашего проекта. Откройте файл `build.gradle` и напишите:

```

// Удаляем mockRelease, так как он не нужен.
android.variantFilter { variant -
    if (variant.buildType.name.equals('release') && variant.getFlavors().get(0).
        name.equals('mock')) {
        variant.setIgnore(true);
    }
}

```

## 47.15. Включение поддержки экспериментальных плагинов NDK для Gradle и AndroidStudio

Включим и настроим экспериментальный плагин Gradle для улучшения поддержки NDK в AndroidStudio.

Убедитесь в соответствии следующим требованиям:

- Gradle 2.10 (для данного примера);
- Android NDK r10 или более поздней версии;
- Android SDK с инструментами сборки v19.0.0 или более поздней версии.

## Настройка файла `MyApp/build.gradle`

Отредактируйте строку `dependencies.classpath` в `build.gradle`, например, замените:

```
classpath 'com.android.tools.build:gradle:2.1.2'
```

на:

```
classpath 'com.android.tools.build:gradle-experimental:0.7.2'
```

(Проверьте последнюю версию самостоятельно и соответствующим образом адаптируйте свою строку, заменив 0.7.2 на номер последней версии).

Файл `build.gradle` должен выглядеть примерно так:

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle-experimental:0.7.2'
    }
}
allprojects {
    repositories {
        jcenter()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

## Настройка файла `MyApp/app/build.gradle`:

Отредактируйте файл `build.gradle`, чтобы он выглядел примерно так, как показано в следующем примере. Номера версий могут выглядеть иначе.

```
apply plugin: 'com.android.model.application'

model {
    android {
        compileSdkVersion 19
        buildToolsVersion "24.0.1"

        defaultConfig {
            applicationId "com.example.mydomain.myapp"
            minSdkVersion.apiLevel 19
            targetSdkVersion.apiLevel 19
            versionCode 1
            versionName "1.0"
        }
        buildTypes {
            release {
                minifyEnabled false
                proguardFiles.add(file('proguard-android.txt'))
            }
        }
        ndk {
            moduleName "myLib"
        }
    }
}

/* В следующих строках приведены примеры некоторых необязательных флагов,
   которые можно установить для настройки среды сборки.
```

```

    */
    cppFlags.add("-I${file("path/to/my/includes/dir")}".toString())
    cppFlags.add("-std=c++11")
    ldLibs.addAll(['log', 'm'])
    stl = "c++_static"
    abiFilters.add("armeabi-v7a")
}
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
}

```

Перед продолжением работы синхронизируйте и проверьте отсутствие ошибок в файлах Gradle.

### Проверка того, включен ли плагин

Сначала убедитесь, что вы загрузили модуль Android NDK. Затем создайте новое приложение в AndroidStudio и добавьте в файл ActivityMain следующее:

```

public class MainActivity implements Activity {
    onCreate() {
        // Предварительно сгенерированный код (не имеет значения, какой)
    }
    static {
        System.loadLibrary("myLib");
    }
    public static native String getString();
}

```

Часть `getString()` будет выделена красным цветом, а это говорит о том, что соответствующая JNI-функция не найдена. Наведите курсор мыши на вызов функции, пока не появится красная лампочка. Щелкните на лампочке и выберите `create function JNI_`. В результате в каталоге `myApp/app/src/main/jni` должен появиться файл `myLib.c`, содержащий правильный вызов JNI-функции. Он должен выглядеть примерно так:

```

#include <jni.h>

JNIEXPORT jstring JNICALL Java_com_example_mydomain_myapp_MainActivity_getString(JNIEnv *env, jobject instance)
{
    // TODO
    return (*env)->NewStringUTF(env, returnValue);
}

```

Если он выглядит иначе, значит, плагин настроен неправильно или не загружен NDK.

## 47.16. Отображение информации о подписи

В некоторых случаях (например, при получении ключа Google API) требуется найти отпечаток ключа хранилища (keystore fingerprint). В Gradle есть удобная задача, которая выводит всю информацию о подписи, включая отпечатки keystore:

```
./gradlew signingReport
```

Пример вывода:

```
:app:signingReport
Variant: release
Config: none
-----
Variant: debug
Config: debug
Store: /Users/user/.android/debug.keystore
Alias: AndroidDebugKey
MD5: 25:08:76:A9:7C:0C:19:35:99:02:7B:00:AA:1E:49:CA
SHA1: 26:BE:89:58:00:8C:5A:7D:A3:A9:D3:60:4A:30:53:7A:3D:4E:05:55
Valid until: Saturday 18 June 2044
-----
Variant: debugAndroidTest
Config: debug
Store: /Users/user/.android/debug.keystore
Alias: AndroidDebugKey
MD5: 25:08:76:A9:7C:0C:19:35:99:02:7B:00:AA:1E:49:CA
SHA1: 26:BE:89:58:00:8C:5A:7D:A3:A9:D3:60:4A:30:53:7A:3D:4E:05:55
Valid until: Saturday 18 June 2044
-----
Variant: debugUnitTest
Config: debug
Store: /Users/user/.android/debug.keystore
Alias: AndroidDebugKey
MD5: 25:08:76:A9:7C:0C:19:35:99:02:7B:00:AA:1E:49:CA
SHA1: 26:BE:89:58:00:8C:5A:7D:A3:A9:D3:60:4A:30:53:7A:3D:4E:05:55
Valid until: Saturday 18 June 2044
-----
Variant: releaseUnitTest
Config: none
```

## 47.17. Просмотр дерева зависимостей

Используйте зависимости задачи. В зависимости от того, как настроены ваши модули, это могут быть либо `./gradlew dependencies`, либо для просмотра зависимостей модуля `app` используйте `./gradlew :app:dependencies`.

Из следующего примера файла `build.gradle`:

```
dependencies {
    compile 'com.android.support:design:23.2.1'
    compile 'com.android.support:cardview-v7:23.1.1'

    compile 'com.google.android.gms:play-services:6.5.87'
}
```

будет получен следующий график:

```
Parallel execution is an incubating feature.
:app:dependencies
-----
Project :app
...
_releaseApk - ## Internal use, do not manually configure ##
```

```

+--- com.android.support:design:23.2.1
| +--- com.android.support:support-v4:23.2.1
| | \--- com.android.support:support-annotations:23.2.1
+--- com.android.support:appcompat-v7:23.2.1
| +--- com.android.support:support-v4:23.2.1 (*)
| +--- com.android.support:animated-vector-drawable:23.2.1
| | \--- com.android.support:support-vector-drawable:23.2.1
| | \--- com.android.support:support-v4:23.2.1 (*)
| +--- com.android.support:support-vector-drawable:23.2.1 (*)
\--- com.android.support:recyclerview-v7:23.2.1
+--- com.android.support:support-v4:23.2.1 (*)
\--- com.android.support:support-annotations:23.2.1
+--- com.android.support:cardview-v7:23.1.1
\--- com.google.android.gms:play-services:6.5.87
\--- com.android.support:support-v4:21.0.0 -> 23.2.1 (*)
...

```

Здесь видно, что проект напрямую включает `com.android.support:design` версии 23.2.1, который сам подтягивает `com.android.support:support-v4` с версией 23.2.1. Однако сам `com.google.android.gms:play-services` имеет зависимость от того же `support-v4`, но со старой версией 21.0.0, что является конфликтом, обнаруженым gradle.

Символы (\*) используются, когда gradle пропускает поддерево, поскольку эти зависимости уже были перечислены ранее.

## 47.18. Отключение сжатия изображений для уменьшения размера APK-файла

Если вы оптимизируете все изображения вручную, отключите APT Cruncher, уменьшающий размер APK-файла.

```

android {
    aaptOptions {
        cruncherEnabled = false
    }
}

```

## 47.19. Автоматическое удаление несогласованных (unaligned) apk

Если вам не нужны автоматически генерируемые apk-файлы с суффиксом `unaligned` («несогласованные»), вы можете добавить следующий код в файл `build.gradle`:

```

// удаление unaligned-файлов
android.applicationVariants.all { variant ->
    variant.assemble.doLast {
        variant.outputs.each { output ->
            println "aligned " + output.outputFile
            println "unaligned " + output.packageApplication.outputFile

            File unaligned = output.packageApplication.outputFile;
            File aligned = output.outputFile
            if (!unaligned.getName().equalsIgnoreCase(aligned.getName()))
                { println "удаление " + unaligned.getName()
                  unaligned.delete()
                }
        }
    }
}

```

## 47.20. Выполнение сценария оболочки из gradle

Сценарий (скрипт) оболочки – это универсальный способ расширить сборку. В качестве примера приведем простой скрипт для компиляции файлов protobuf и добавления получившихся java-файлов в каталог исходных текстов для дальнейшей компиляции:

```
def compilePb() {
    exec {
        // ВНИМАНИЕ: gradle потерпит неудачу, если в файле protoc есть ошибка
        executable "../pbScript.sh"
    }
}

project.afterEvaluate {
    compilePb()
}
```

Сценарий (скрипт) оболочки ‘pbScript.sh’ для данного примера, расположенный в корневой папке проекта:

```
#!/usr/bin/env bash
pp=/home/myself/my/proto

/usr/local/bin/protoc -I=$pp \
--java_out=./src/main/java \
--proto_path=$pp \
$pp/my.proto \
--proto_path=$pp \
$pp/my_other.proto
```

## 47.21. Как показать все задачи проекта gradle

```
gradlew tasks -- show all tasks
```

Android tasks (Задачи Android)

-----  
androidDependencies - Displays the Android dependencies of the project (Отображает Android-зависимости проекта.)

signingReport - Displays the signing info for each variant. (Отображает информацию о подписи для каждого варианта.)

sourceSets - Prints out all the source sets defined in this project. (Выводит все исходные наборы, определенные в данном проекте.)

Build tasks (Задачи сборки)

-----  
assemble - Assembles all variants of all applications and secondary packages. (Сборка всех вариантов всех приложений и вторичных пакетов.)

assembleAndroidTest - Assembles all the Test applications. (Сборка всех тестовых приложений.)

assembleDebug - Assembles all Debug builds. (Сборка всех отладочных сборок)

assembleRelease - Assembles all Release builds. (Сборка всех релизных сборок.)

build - Assembles and tests this project. (Сборка и тестирование данного проекта.)

buildDependents - Assembles and tests this project and all projects that depend on it. (Сборка и тестирование данного проекта и всех проектов, зависящих от него.)

buildNeeded - Assembles and tests this project and all projects it depends on. (Собирает и тестирует данный проект и все проекты, от которых он зависит.)

classes - Assembles main classes. (Собирает основные классы.)

clean - Deletes the build directory. (Удаляет каталог сборки.)

compileDebugAndroidTestSources

compileDebugSources  
compileDebugUnitTestSources  
compileReleaseSources  
compileReleaseUnitTestSources  
extractDebugAnnotations - Extracts Android annotations for the debug variant into the archive file. (Извлечение в архивный файл аннотаций Android для отладочного варианта.)  
extractReleaseAnnotations - Extracts Android annotations for the release variant into the archive file. (Извлечение в архивный файл аннотаций Android для релизного варианта.)  
jar - Assembles a jar archive containing the main classes. (Сборка jar-архива, содержащего основные классы.)  
mockableAndroidJar - Creates a version of android.jar that is suitable for unit tests. (Создает версию android.jar, подходящую для модульных тестов.)  
testClasses - Assembles test classes. (Собирает тестовые классы.)

#### Build Setup tasks (Задачи настройки сборки)

-----  
init - Initializes a new Gradle build. [incubating] (Инициализирует новую сборку Gradle.)  
wrapper - Generates Gradle wrapper files. [incubating] (Генерирует файлы обёртки Gradle.)

#### Documentation tasks (Задачи документирования)

-----  
javadoc - Generates Javadoc API documentation for the main source code. (Генерирует документацию Javadoc API для основного исходного кода.)

#### Help tasks (Задачи помощи)

-----  
buildEnvironment - Displays all buildscript dependencies declared in root project 'LeitnerBoxPro'. (Отображает все зависимости buildscript, объявленные в корневом проекте.)  
components - Displays the components produced by root project 'LeitnerBoxPro'. [incubating] (Отображает компоненты, произведенные корневым проектом.)  
dependencies - Displays all dependencies declared in root project 'LeitnerBoxPro'. (Отображает все зависимости, объявленные в корневом проекте.)  
dependencyInsight - Displays the insight into a specific dependency in root project 'LeitnerBoxPro'. (Отображает информацию о конкретной зависимости в корневом проекте.)  
help - Displays a help message. (Отображение справочного сообщения.)  
model - Displays the configuration model of root project 'LeitnerBoxPro'. [incubating] (Отображает модель конфигурации корневого проекта.)  
projects - Displays the sub-projects of root project 'LeitnerBoxPro'. (Отображает подпроекты корневого проекта.)  
properties - Displays the properties of root project 'LeitnerBoxPro'. (Отображает свойства корневого проекта.)  
tasks - Displays the tasks runnable from root project 'LeitnerBoxPro' (some of the displayed tasks may belong to subprojects). (Отображает задачи, выполняемые из корневого проекта (некоторые из отображаемых задач могут принадлежать подпроектам).)

#### Install tasks (Задачи установки)

-----  
installDebug - Installs the Debug build. (Устанавливает отладочную сборку.)  
installDebugAndroidTest - Installs the android (on device) tests for the Debug build. (Устанавливает тесты android (на устройстве) для отладочной сборки.)  
uninstallAll - Uninstall all applications. (Удаление всех приложений.)  
uninstallDebug - Uninstalls the Debug build. (Удаление отладочной сборки.)  
uninstallDebugAndroidTest - Uninstalls the android (on device) tests for the Debug

`build.` (Удаление тестов android (на устройстве) для отладочной сборки.)  
`uninstallRelease` - Uninstalls the Release build. (Удаление сборки Release.)

#### Verification tasks (Задачи верификации)

`check` - Runs all checks. (Выполняет все проверки.)  
`connectedAndroidTest` - Installs and runs instrumentation tests for all flavors on connected devices. (Устанавливает и запускает инструментальные тесты для всех вариантов на подключенных устройствах.)  
`connectedCheck` - Runs all device checks on currently connected devices.  
`connectedDebugAndroidTest` - Installs and runs the tests for debug on connected devices. (Выполняет все проверки устройств на подключенных в данный момент устройствах.)  
`deviceAndroidTest` - Installs and runs instrumentation tests using all Device Providers. (Устанавливает и запускает тесты для отладки с использованием всех Device Providers.)  
`deviceCheck` - Runs all device checks using Device Providers and Test Servers. (Выполняет все проверки устройств с использованием Device Providers и Test Servers.)  
`lint` - Runs lint on all variants. (Выполняет проверку всех вариантов.)  
`lintDebug` - Runs lint on the Debug build. (Выполняет проверку отладочной сборки.)  
`lintRelease` - Runs lint on the Release build. (Запуск проверки на сборке Release)  
`test` - Run unit tests for all variants. (Запуск модульных тестов для всех вариантов.)  
`testDebugUnitTest` - Run unit tests for the debug build. (Запуск модульных тестов для отладочной сборки.)  
`testReleaseUnitTest` - Run unit tests for the release build. (Запуск модульных тестов для сборки релиза.)

#### Other tasks

`assembleDefault`

`clean`

`jarDebugClasses`

`jarReleaseClasses`

`transformResourcesWithMergeJavaResForDebugUnitTest`

`transformResourcesWithMergeJavaResForReleaseUnitTest`

## 47.22. Отладка ошибок в Gradle

Допустим, вы разрабатываете приложение и получаете некоторую ошибку Gradle, которая в общем случае выглядит следующим образом:

```
:module:someTask FAILED
FAILURE: Build failed with an exception.
* What went wrong:
Execution failed for task ':module:someTask'.
> some message here... finished with non-zero exit value X
* Try:
Run with --stacktrace option to get the stack trace. Run with --info or --debug
option to get more
log output.
BUILD FAILED
Total time: Y.ZZ secs
```

Вы ищете свою проблему на StackOverflow, и другие разработчики пишут, что нужно почистить и пересобрать проект или включить MultiDex, но когда вы пробуете это сделать, проблема не решается.

Сам вывод Gradle должен указывать на фактическую ошибку в нескольких строках выше сообщения между :module:someTask FAILED и последним прошедшим :module:someOtherTask. Поэтому, если вы задаете вопрос о своей ошибке, пожалуйста, отредактируйте свои вопросы, включив в них дополнительный контекст.

Итак, вы получаете «ненулевое значение выхода» (non-zero exit value). Это число – хороший индикатор того, что следует попытаться исправить. Вот несколько наиболее часто встречающихся вариантов.

- 1 – просто общий код ошибки, и ошибка, скорее всего, находится в выводе Gradle.
- 2 – обычно связано с пересекающимися (overlapping) зависимостями или неправильной конфигурацией проекта.
- 3 – обычно связано с включением слишком большого количества зависимостей или с проблемой памяти.

Общими решениями для вышеописанных ситуаций (после попытки очистки и пересборки проекта) являются следующие:

- при варианте 1 – устранить указанную ошибку. Как правило, это ошибка времени компиляции, означающая, что какой-то фрагмент кода в вашем проекте не является корректным. Это относится как к XML, так и к Java для проекта Android;
- при вариантах 2 и 3 часто предлагается включить multidex (<http://developer.android.com/tools/building/multidex.html>). Хотя это может решить проблему, скорее всего, это обходной путь. Если вы не понимаете, зачем он нужен (см. ссылку), то, скорее всего, он вам не нужен. Общие решения включают в себя сокращение чрезмерного использования зависимостей от библиотек (например, всех служб Google Play Services, когда необходимо использовать только одну библиотеку, например, Maps или Sign-In).

## 47.23. Использование gradle.properties для централизованного определения номера версии или конфигурации сборки

Вы можете определить информацию о центральной конфигурации в:

- отдельном включаемом файле gradle, централизуя зависимости с помощью файла "dependencies.gradle";
- отдельном файле свойств, управляя версиями сборок с помощью файла "version.properties".

...или сделать это с помощью корневого файла gradle.properties.

Структура проекта:

```
root
+- module1/
|   build.gradle
+- модуль2/
|   build.gradle
+- build.gradle
+- gradle.properties
```

Глобальная настройка для всех подмодулей в файле gradle.properties:

```
# used for manifest
# todo increment for every release
appVersionCode=19
appVersionName=0.5.2.160726
# android tools settings
```

```
appCompileSdkVersion=23
appBuildToolsVersion=23.0.2
```

Использование в субмодуле:

```
apply plugin: 'com.android.application'
android {
    // appXXX определяются в gradle.properties
    compileSdkVersion = Integer.valueOf(appCompileSdkVersion)
    buildToolsVersion = appBuildToolsVersion

    defaultConfig {
        // appXXX определяются в файле gradle.properties
        versionCode = Long.valueOf(appVersionCode)
        versionName = appVersionName
    }
}

dependencies {
    ...
}
```

## 47.24. Определение типов сборки

Создавать и настраивать типы сборки можно в файле `build.gradle` на уровне модуля внутри блока `android {}`.

```
android {
    ...
    defaultConfig {...}

    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
            'proguard-rules.pro'
        }

        debug {
            applicationIdSuffix ".debug"
        }
    }
}
```

# Глава 48. Файловый ввод/вывод в Android

Чтение и запись файлов в Android не отличаются от стандартных чтения и записи файлов в Java. Можно использовать тот же пакет `java.io`. Однако есть некоторая специфика, связанная с каталогами, в которые разрешено писать, разрешениями в целом и обходными путями МТР.

## 48.1. Получение рабочего каталога

Получить рабочий каталог можно, вызвав метод `getFilesDir()` в активности `Activity` (`Activity` – центральный класс приложения, наследующий от `Context`). Чтение ничем не отличается; доступ к этой папке будет иметь только ваше приложение.

Ваша активность может содержать, например, следующий код:

```
File myFolder = getFilesDir();
File myFile = new File(myFolder, "myData.bin");
```

## 48.2. Запись необработанного массива байтов

```
File myFile = new File(getFilesDir(), "myData.bin");
FileOutputStream out = new FileOutputStream(myFile);
```

```
// Запись четырех байтов (один два три четыре):
out.write(new byte[] {1, 2, 3, 4})
out.close()
```

В этом коде нет ничего специфичного для Android. Если вы часто записываете много маленьких значений, используйте `BufferedOutputStream` (см. <https://developer.android.com/reference/java/io/BufferedOutputStream.html>), чтобы уменьшить износ внутреннего SSD-накопителя устройства.

## 48.3. Сериализация объекта

Старая добрая сериализация объектов Java доступна и в Android. Вы можете определить `Serializable`-классы, например:

```
class Circle implements Serializable {
    final int radius;
    final String name;

    Circle(int radius, int name) {
        this.radius = radius;
        this.name = name;
    }
}
```

...и затем записать в поток `ObjectOutputStream`:

```
File myFile = new File(getFilesDir(), "myObjects.bin");
FileOutputStream out = new FileOutputStream(myFile);
ObjectOutputStream oout = new ObjectOutputStream(new BufferedOutputStream(out));

oout.writeObject(new Circle(10, "One"));
oout.writeObject(new Circle(12, "Two"));

oout.close()
```

Сериализация объектов Java может быть как идеальным, так и очень плохим выбором, в зависимости от того, что вы хотите сделать с ее помощью. Если вы решите использовать версионирование, сначала прочитайте о нем (<http://www.javaworld.com/article/2071731/core-java/ensure-proper-version-control-for-serialized-objects.html>).

## 48.4. Запись на внешний носитель (SD-карту)

Также можно производить чтение и запись с/на карту памяти (SD-карту), которая присутствует во многих устройствах Android. Доступ к файлам на ней могут получить другие про-

грамм, а также непосредственно пользователь после подключения устройства к компьютеру через USB-кабель и включения протокола MTP.

Определение местоположения SD-карты несколько более проблематично. Класс `Environment` содержит статические методы для получения "внешних каталогов", которые обычно должны находиться на SD-карте, а также информацию о том, существует ли SD-карта вообще и доступна ли она для записи. Дополнительно об этом можно почитать на ресурсе <http://stackoverflow.com/questions/5694933/find-an-external-sd-card-location>.

Для доступа к внешнему хранилищу требуются разрешения в манифесте Android:

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

В старых версиях Android для установки разрешений достаточно поместить их в манифест (пользователь должен одобрить их при установке). Однако начиная с Android 6.0 у пользователя запрашивается разрешение при первом доступе. В противном случае доступ будет запрещен независимо от вашего манифеста.

## 48.5. Решение проблемы с невидимыми файлами MTP

При создании файлов для экспорта через USB-кабель на настольный компьютер по протоколу MTP может возникнуть проблема: вновь созданные файлы не сразу видны в файловом проводнике, запущенном на подключенном настольном компьютере. Чтобы сделать новые файлы видимыми, необходимо вызвать `MediaScannerConnection` (<https://developer.android.com/reference/android/media/MediaScannerConnection.html>):

```
File file = new File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMENTS), "theDocument.txt");
FileOutputStream out = new FileOutputStream(file)

... (запись документа)

out.close()
MediaScannerConnection.scanFile(this, new String[] {file.getPath()}, null, null);
context.sendBroadcast(new Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE, Uri.fromFile(file)));
```

Данный код вызова `MediaScannerConnection` работает только для файлов, но не для каталогов.

## 48.6. Работа с большими файлами

Небольшие файлы обрабатываются за доли секунды, и вы можете читать или записывать их вместо кода, где это необходимо. Однако если файл больше или обрабатывается медленнее, то для работы с ним в фоновом режиме может потребоваться использование `AsyncTask` в Android:

```
class FileOperation extends AsyncTask<String, Void, File> {

    @Override
    protected File doInBackground(String... params) {
        try {
            File file = new File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMENTS), "bigAndComplexDocument.odf");
            FileOutputStream out = new FileOutputStream(file)

            ... (запись документа)
```