

Создание обработчика для текущего потока:

```
Handler handler = new Handler();
```

Создание обработчика для основного потока (UI Thread):

```
Handler handler = new Handler(Looper.getMainLooper());
```

Передача Runnable из другого потока в основной поток:

```
new Thread(new Runnable() {
    public void run() {
        // это выполняется на другом потоке

        // создаем обработчик, связанный с основным потоком
        Handler handler = new Handler(Looper.getMainLooper());

        // отправим Runnable в основной поток
        handler.post(new Runnable() {
            public void run() {
                // это выполняется на основном потоке
            }
        });
    }
}).start();
```

Создание обработчика для другого потока HandlerThread и отправка ему событий:

```
// создадим еще один поток
HandlerThread otherThread = new HandlerThread("name");

// создадим обработчик, связанный с другим потоком
Handler handler = new Handler(otherThread.getLooper());

// отправим событие другому потоку
handler.post(new Runnable() {
    public void run() {
        // это выполняется на другом потоке
    }
});
```

71.2. Использование обработчика для создания таймера (аналогично javax.swing.Timer)

Это может быть полезно, если вы пишете игру или что-то, что должно выполнять фрагмент кода каждые несколько секунд:

```
import android.os.Handler;

public class Timer {
    private Handler handler;
    private boolean paused;

    private int interval;

    private Runnable task = new Runnable () {
        @Override
```

```

public void run() {
    if (!paused) {
        runnable.run();
        Timer.this.handler.postDelayed (this, interval);
    }
}
};

private Runnable runnable;

public int getInterval() {
    return interval;
}

public void setInterval(int interval) {
    this.interval = interval;
}

public void startTimer () {
    paused = false;
    handler.postDelayed (task, interval);
}

public void stopTimer () {
    paused = true;
}

public Timer (Runnable runnable, int interval, boolean started) {
    handler = new Handler ();
    this.runnable = runnable;
    this.interval = interval;
    if (started)
        startTimer ();
}
}

```

Пример использования:

```

Timer timer = new Timer(new Runnable() {
    public void run() {
        System.out.println("Hello");
    }
}, 1000, true)

```

Этот код будет печатать "Hello" каждую секунду.

71.3. Использование обработчика для выполнения кода через промежуток времени

Выполнение кода через 1,5 секунды:

```

Handler handler = new Handler();
handler.postDelayed(new Runnable() {
    @Override
    public void run() {
        //Код, который необходимо выполнить по истечении времени
    }
}, 1500)

```

```

    }
}; 1500); //время задержки в миллисекундах

```

Повторное выполнение кода каждую секунду:

```

Handler handler = new Handler();
handler.postDelayed(new Runnable() {
    @Override
    public void run() {
        handler.postDelayed(this, 1000);
    }
}, 1000); //время задержки в миллисекундах

```

71.4. Остановка выполнения обработчика

Чтобы остановить выполнение обработчика, удалите привязанный к нему обратный вызов с помощью запущенного внутри него runnable:

```

Runnable my_runnable = new Runnable() {
    @Override
    public void run() {
        // ваш код здесь
    }
};

public Handler handler = new Handler(); // используйте 'new Handler(Looper.getMainLooper())', если вы хотите, чтобы этот обработчик управлял чем-то в пользовательском интерфейсе
// для запуска обработчика
public void start() {
    handler.postDelayed(my_runnable, 10000);
}

// для остановки обработчика
public void stop() {
    handler.removeCallbacks(my_runnable);
}

// для сброса обработчика
public void restart() {
    handler.removeCallbacks(my_runnable);
    handler.postDelayed(my_runnable, 10000);
}

```

Глава 72. BroadcastReceiver

BroadcastReceiver (широковещательный приемник) – это компонент Android, позволяющий регистрироваться для получения системных или прикладных событий. Все зарегистрированные приемники события получают уведомления от среды выполнения Android, как только это событие происходит. Например, сообщение о выключении экрана, разряде батареи или о том, что был сделан снимок. Приложения также могут инициировать **широковещательные рассылки (broadcasts)** – например, чтобы сообщить другим приложениям о том, что некоторые данные загружены на устройство и доступны для использования.

72.1. Использование LocalBroadcastManager

LocalBroadcastManager используется для передачи широковещательных намерений (Broadcast Intents) внутри приложения, не раскрывая их нежелательным слушателям.

Использование LocalBroadcastManager более эффективно и безопасно, чем прямое использование context.sendBroadcast(), поскольку не нужно беспокоиться о подделке широковещательных сообщений другими приложениями, что может представлять угрозу безопасности.

Приведем простой пример отправки и приема локальных широковещательных сообщений:

```
BroadcastReceiver receiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals("Некое действие")) {
            // Выполнить что-либо
        }
    }
});

LocalBroadcastManager manager = LocalBroadcastManager.getInstance(mContext);
manager.registerReceiver(receiver, new IntentFilter("Некое действие"));

// в результате этого вызова будет вызвана функция onReceive():
manager.sendBroadcast(new Intent("Некое действие")); //См. также sendBroadcastSync

// Не забудьте снять приемник с регистрации, когда закончите работу с ним:
manager.unregisterReceiver(receiver);
```

72.2. Основы BroadcastReceiver

BroadcastReceivers используются для приема широковещательных намерений, посылаемых ОС Android, другими приложениями или внутри одного приложения.

Для создания каждого намерения (Intent) используется *фильтр интентов (намерений)* (Intent Filter), который требует *действия* типа String. Дополнительная информация может быть настроена в намерении.

Аналогичным образом широковещательные приемники регистрируются для получения интентов с определенным фильтром интентов. Они могут быть зарегистрированы программно:

```
mContext.registerReceiver(new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        //Ваша реализация находится здесь.
    }
}, new IntentFilter("Some Action"));
```

...или в файле AndroidManifest.xml:

```
<receiver android:name=".MyBroadcastReceiver">
    <intent-filter>
        <action android:name="Some Action"/>
    </intent-filter>
</receiver>
```

Чтобы получить намерение, установите Action на что-то документированное ОС Android, другим приложением или API, либо внутри собственного приложения, используя sendBroadcast:

```
mContext.sendBroadcast(new Intent("Some Action"));
```

72.3. Дополнительная информация о приемниках вещания

Кроме того, намерение может содержать информацию, такую как строки, примитивы и Parcelable-объекты, которые можно просмотреть в методе `onReceive()`.

72.3. Дополнительная информация о приемниках вещания

Широковещательный приемник – компонент Android, позволяющий регистрировать системные или прикладные события, – может быть зарегистрирован через файл `AndroidManifest.xml` или динамически с помощью метода `Context.registerReceiver()`:

```
public class MyReceiver extends BroadcastReceiver {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        //Ваша реализация находится здесь.  
    }  
}
```

Для примера рассмотрим `ACTION_BOOT_COMPLETED`, который запускается системой после завершения процесса загрузки Android.

Зарегистрировать приемник в файле манифеста можно следующим образом:

```
<application  
    android:icon="@drawable/ic_launcher"  
    android:label="@string/app_name"  
    android:theme="@style/AppTheme" >  
    <receiver android:name="MyReceiver">  
        <intent-filter>  
            <action android:name="android.intent.action.BOOT_COMPLETED">  
            </action>  
        </intent-filter>  
    </receiver>  
</application>
```

Теперь устройство загрузится, будет вызван метод `onReceive()`, после чего можно выполнять свои действия (например, запустить службу либо alarm-сигнал).

72.4. Использование упорядоченных трансляций

Упорядоченные трансляции (Ordered broadcasts) используются в тех случаях, когда необходимо указать приоритет для слушателей передачи.

В данном примере первый получатель (`firstReceiver`) будет всегда получать широковещательную рассылку раньше, чем второй (`secondReceiver`):

```
final int highPriority = 2;  
final int lowPriority = 1;  
final String action = "действие";  
  
// фильтр намерений (intent filter) для первого получателя с высоким приоритетом  
final IntentFilter firstFilter = new IntentFilter(action);  
firstFilter.setPriority(highPriority);  
final BroadcastReceiver firstReceiver = new MyReceiver();  
  
// фильтр намерений для второго приемника с низким приоритетом  
final IntentFilter secondFilter = new IntentFilter(action);  
secondFilter.setPriority(lowPriority);  
final BroadcastReceiver secondReceiver = new MyReceiver();
```

```
// регистрируем приемники
context.registerReceiver(firstReceiver, firstFilter);
context.registerReceiver(secondReceiver, secondFilter);

// отправка упорядоченной широковещательной рассылки
context.sendOrderedBroadcast(new Intent(action), null);
```

Кроме того, приемник широковещания может прервать заказанную передачу:

```
@Override
public void onReceive(final Context context, final Intent intent) {
    abortBroadcast();
}
```

В этом случае все приемники с более низким приоритетом не получат широковещательное сообщение.

72.5. Липкая трансляция (Sticky Broadcast)

Если мы используем метод `sendStickyBroadcast(intent)`, то соответствующее намерение является «липким» (sticky), то есть отправляемое намерение остается после завершения трансляции. StickyBroadcast, как следует из названия, это механизм чтения данных из трансляции после ее завершения. Это может быть использовано в случае, когда необходимо проверить, например, в методе `onCreate()` активности значение ключа в намерении до того, как эта активность была запущена.

```
Intent intent = new Intent("com.org.action");
intent.putExtra("anIntegerKey", 0);
sendStickyBroadcast(intent);
```

72.6. Включение и отключение приемника вещания программным способом

Чтобы включить или отключить `BroadcastReceiver`, нам необходимо получить ссылку на `PackageManager`, а для этого нужен объект `ComponentName`, содержащий класс приемника, который мы хотим включить или выключить:

```
ComponentName componentName = new ComponentName(context, MyBroadcastReceiver.class);
PackageManager packageManager = context.getPackageManager();
```

Теперь мы можем вызвать следующий метод для включения `BroadcastReceiver`:

```
packageManager.setComponentEnabledSetting(
    componentName,
    PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
    PackageManager.DONT_KILL_APP);
```

...или мы можем вместо этого использовать `COMPONENT_ENABLED_STATE_DISABLED` для отключения приемника:

```
packageManager.setComponentEnabledSetting(
    componentName,
    PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
    PackageManager.DONT_KILL_APP);
```

72.7. Пример LocalBroadcastManager

`BroadcastReceiver` – это, по сути, механизм передачи намерений через ОС для выполнения определенных действий. Классическое определение:

Broadcast receiver – это компонент Android, позволяющий регистрировать события системы или приложения.

LocalBroadcastManager – это способ отправки или получения широковещательных сообщений внутри процесса приложения. Этот механизм обладает целым рядом преимуществ:

1. Поскольку данные остаются внутри процесса приложения, их утечка невозможна.
 2. LocalBroadcasts разрешаются быстрее, чем обычные широковещательные сообщения.
- Простым примером LocalBroadcastManager является:

SenderIdActivity

```
Intent intent = new Intent("anEvent");
intent.putExtra("key", "This is an event");
LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
```

ReceiverActivity

1. Зарегистрируйте приемник:

```
LocalBroadcastManager.getInstance(this).registerReceiver(aLBReceiver, new
IntentFilter("anEvent"));
```

2. Конкретный объект для выполнения действий при вызове приемника:

```
private BroadcastReceiver aLBReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        // выполнить действие здесь.
    }
};
```

3. Снятие с регистрации, когда вид больше не используется.

```
@Override
protected void onPause() {
    // Снять с регистрации, так как активность скоро будет закрыта.
    LocalBroadcastManager.getInstance(this).unregisterReceiver(aLBReceiver);
    super.onDestroy();
}
```

72.8. Остановленное состояние (stopped state)

Начиная с версии Android 3.1 все приложения после установки переводятся в остановленное состояние, или состояние остановки (stopped state). В остановленном состоянии приложение не запускается ни по какой причине, кроме как при ручном запуске активности или при явном намерении, адресованном активности, службе или широковещательной рассылке.

При написании системного приложения, устанавливающего APK напрямую, следует учитывать, что только что установленное приложение не будет получать никаких трансляций до тех пор, пока не выйдет из состояния остановки.

Простым способом активации приложения является отправка ему явного широковещательного сообщения. Большинство приложений реализуют INSTALL_REFERRER, мы можем использовать его в качестве точки привязки.

Просканируйте манифест установленного приложения и отправьте явную широковещательную рассылку каждому получателю:

```
Intent intent = new Intent();
intent.addFlags(Intent.FLAG_INCLUDE_STOPPED_PACKAGES);
intent.setComponent(new ComponentName(packageName, fullClassName));
sendBroadcast(intent);
```

72.9. Передача данных между двумя активностями через пользовательский BroadcastReceiver

Можно установить связь между двумя активностями, чтобы активность А была уведомлена о событии, происходящем в активности В.

Активность А:

```
final String eventName = "your.package.goes.here.EVENT";

@Override
protected void onCreate(Bundle savedInstanceState) {
    registerEventReceiver();
    super.onCreate(savedInstanceState);
}

@Override
protected void onDestroy() {
    unregisterEventReceiver(eventReceiver);
    super.onDestroy();
}

private void registerEventReceiver() {
    IntentFilter eventFilter = new IntentFilter();
    eventFilter.addAction(eventName);
    registerReceiver(eventReceiver, eventFilter);
}

private BroadcastReceiver eventReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        //Этот код будет выполнен, когда будет запущена трансляция в активности В
    }
};
```

Активность В:

```
final String eventName = "your.package.goes.here.EVENT";

private void launchEvent() {
    Intent eventIntent = new Intent(eventName);
    this.sendBroadcast(eventIntent);
}
```

Конечно, вы можете добавить больше информации в трансляцию, добавив дополнения к интенту, передаваемому между активностями. Здесь они не добавлены, чтобы максимально упростить пример.

72.10. BroadcastReceiver для обработки событий BOOT_COMPLETED

В примере ниже показано, как создать BroadcastReceiver, способный принимать события BOOT_COMPLETED. Таким образом, вы сможете запустить службу или активность сразу после включения устройства.

Кроме того, для восстановления alarm-сигналов также можно использовать события BOOT_COMPLETED, поскольку при выключении устройства сигналы уничтожаются.

Примечание: пользователь должен хотя бы один раз запустить приложение, прежде чем можно будет получить действие BOOT_COMPLETED.

AndroidManifest.xml

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.test.example" >
    ...
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    ...
    <application>
        ...
        <receiver android:name="com.test.example.MyCustomBroadcastReceiver">
            <intent-filter>
                <!-- РЕГИСТРИРУЙТЕСЬ ДЛЯ ПОЛУЧЕНИЯ СОБЫТИЙ BOOT_COMPLETED -->
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>
    </application>
</manifest>

```

MyCustomBroadcastReceiver.java

```

public class MyCustomBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if(action != null) {
            if (action.equals(Intent.ACTION_BOOT_COMPLETED) ) {
                // TODO: Код для обработки события BOOT_COMPLETED
                // TODO: запуск службы... отображение уведомления... запуск активности
            }
        }
    }
}

```

72.11. Bluetooth BroadcastReceiver

Добавим разрешение в файл манифеста:

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

В вашем фрагменте (или активности):

- Добавьте метод приемника:

```

private BroadcastReceiver mBluetoothStatusChangedReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final Bundle extras = intent.getExtras();
        final int bluetoothState = extras.getInt(Constants.BUNDLE_BLUETOOTH_STATE);
        switch(bluetoothState) {
            case BluetoothAdapter.STATE_OFF:
                // Bluetooth выключен
                break;
        }
    }
}

```

```

        case BluetoothAdapter.STATE_TURNING_OFF:
            // Выключение
            break;
        case BluetoothAdapter.STATE_ON:
            // Bluetooth включен
            break;
        case BluetoothAdapter.STATE_TURNING_ON:
            // Включение
            break;
    }
};

}

```

Регистрация вещания:

- Вызовите этот метод на onResume():

```

private void registerBroadcastManager(){
    final LocalBroadcastManager manager = LocalBroadcastManager.
        getInstance(getApplicationContext());
    manager.registerReceiver(mBluetoothStatusChangedReceiver, new
        IntentFilter(Constants.BROADCAST_BLUETOOTH_STATE));
}

```

Снять вещание с регистрации:

- Вызовите этот метод на onPause():

```

private void unregisterBroadcastManager(){
    final LocalBroadcastManager manager = LocalBroadcastManager.
        getInstance(getApplicationContext());
    manager.unregisterReceiver(mBluetoothStatusChangedReceiver);
}

```

Глава 73. Жизненный цикл пользовательского интерфейса

73.1. Сохранение данных при обрезке памяти

```

public class ExampleActivity extends Activity {

    private final static String EXAMPLE_ARG = "example_arg";
    private int mArg;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_example);

        if(savedInstanceState != null) {
            mArg = savedInstanceState.getInt(EXAMPLE_ARG);
        }
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {

```

```
super.onSaveInstanceState(outState);
outState.putInt(EXAMPLE_ARG, mArg);
```

Пояснение

Итак, что же здесь происходит?

Система Android всегда будет стремиться освободить как можно больше памяти. Поэтому, если ваша активность уходит в фон, а другая активность переднего плана требует свою долю, система Android вызовет для вашей активности функцию `onTrimMemory()`.

Но это не означает, что все ваши свойства должны исчезнуть. Лучше всего сохранить их в объекте `Bundle` (бандл). Такие объекты гораздо лучше работают с памятью. Внутри бандла каждый объект идентифицируется уникальной текстовой последовательностью – в приведенном примере целочисленная переменная `arg` хранится под ссылочным именем `EXAMPLE_ARG`. При повторном создании активности можно извлечь старые значения из бандла, а не создавать их заново.

Глава 74. HttpURLConnection

74.1. Создание HttpURLConnection

Для того чтобы создать новый Android HTTP-клиент HttpURLConnection, вызовите `openConnection()` на экземпляре URL. Поскольку `openConnection()` возвращает `URLConnection`, необходимо явно привести возвращаемое значение:

```
URL url = new URL("http://example.com");
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
// что-то делать с соединением
```

Если вы создаете новый URL, то вам также необходимо обработать исключения, связанные с разбором URL:

```
try {
    URL url = new URL("http://example.com");
    HttpURLConnection connection = (HttpURLConnection) url.openConnection();
    // что-то делать с соединением
} catch (MalformedURLException e) {
    e.printStackTrace();
}
```

После того как тело ответа прочитано и соединение больше не требуется, его следует закрыть, вызвав `disconnect()`.

Приведем пример:

```
URL url = new URL("http://example.com");
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
try {
    // что-то делать с соединением
} finally {
    connection.disconnect();
}
```

74.2. Отправка запроса HTTP GET

```

URL url = new URL("http://example.com");
HttpURLConnection connection = (HttpURLConnection) url.openConnection();

try {
    BufferedReader br = new BufferedReader(new InputStreamReader(connection.
        getInputStream()));

    // чтение входного потока
    // в данном случае просто считываем первую строку потока
    String line = br.readLine();
    Log.d("HTTP-GET", line);
} finally {
    connection.disconnect();
}

```

Обратите внимание, что в приведенном примере исключения не обрабатываются. Полный пример, включающий (тривиальную) обработку исключений, будет выглядеть следующим образом:

```

URL url;
HttpURLConnection connection = null;
try {
    url = new URL("http://example.com");
    connection = (HttpURLConnection) url.openConnection();
    BufferedReader br = new BufferedReader(new InputStreamReader(connection.
        getInputStream()));

    // чтение входного потока
    // в данном случае просто считываем первую строку потока
    String line = br.readLine();
    Log.d("HTTP-GET", line);

} catch (IOException e) {
    e.printStackTrace();
} finally {
    if (connection != null) {
        connection.disconnect();
    }
}

```

74.3. Чтение тела запроса HTTP GET

```

URL url = new URL("http://example.com");
HttpURLConnection connection = (HttpURLConnection) url.openConnection();

try {
    BufferedReader br = new BufferedReader(new InputStreamReader(connection.
        getInputStream()));

    // использовать построитель строк для буферизации тела ответа
    // чтение из входного потока
    StringBuilder sb = new StringBuilder();
    String line;

```

```

while ((line = br.readLine()) != null) {
    sb.append(line).append('\n');
}

// использовать построитель строк напрямую,
// или преобразовать его в строку
String body = sb.toString();

Log.d("HTTP-GET", body);

} finally {
    connection.disconnect();
}

```

Обратите внимание, что в приведенном примере исключения не обрабатываются.

74.4. Отправка HTTP POST-запроса с параметрами

Используйте `HashMap` для хранения параметров, которые должны быть отправлены на сервер через POST-параметры:

```
HashMap<String, String> params;
```

После того как `HashMap` параметров заполнен, создайте `StringBuilder`, который будет использоваться для их отправки на сервер:

```

StringBuilder sbParams = new StringBuilder();
int i = 0;
for (String key : params.keySet()) {
    try {
        if (i != 0) {
            sbParams.append("&");
        }
        sbParams.append(key).append("=")
            .append(URLEncoder.encode(params.get(key), "UTF-8"));
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    } i++;
}

```

Затем создайте `HttpURLConnection`, откройте соединение и отправьте POST-параметры:

```

try{
    String url = "http://www.example.com/test.php";
    URL urlObj = new URL(url);
    HttpURLConnection conn = (HttpURLConnection)urlObj.openConnection();
    conn.setDoOutput(true);
    conn.setRequestMethod("POST");
    conn.setRequestProperty("Accept-Charset", "UTF-8");

    conn.setReadTimeout(10000);
    conn.setConnectTimeout(15000);

    conn.connect();

    String paramsString = sbParams.toString();

```

```

DataOutputStream wr = new DataOutputStream(conn.getOutputStream());
wr.writeBytes(paramsString);
wr.flush();
wr.close();
} catch (IOException e) {
    e.printStackTrace();
}
}

```

Затем получим результат, который сервер отправляет обратно:

```

try {
    InputStream in = new BufferedInputStream(conn.getInputStream());
    BufferedReader reader = new BufferedReader(new InputStreamReader(in));
    StringBuilder result = new StringBuilder();
    String line;
    while ((line = reader.readLine()) != null) {
        result.append(line);
    }
    Log.d("тестовый", "результат с сервера: " + result.toString());
} catch (IOException e) {
    e.printStackTrace();
} finally {
    if (conn != null) {
        conn.disconnect();
    }
}

```

74.5. Многоцелевой класс HttpURLConnection для обработки всех типов HTTP-запросов

Следующий класс может быть использован как единый класс, который может обрабатывать запросы GET, POST, PUT, PATCH и другие:

```

class APIResponseObject{
    int responseCode;
    String response;
}

APIResponseObject(int responseCode, String response)
{
    this.responseCode = responseCode;
    this.response = response;
}

public class APIAccessTask extends AsyncTask<String,Void,APIResponseObject> {
    URL requestUrl;
    Context context;
    HttpURLConnection urlConnection;
    List<Pair<String, String>> postData, headerData;
    String method;
    int responseCode = HttpURLConnection.HTTP_OK;

    interface OnCompleteListener{
        void onComplete(APIResponseObject result);
    }
}

```

```
public OnCompleteListener delegate = null;

APIAccessTask(Context context, String requestUrl, String method,
OnCompleteListener delegate){
    this.context = context;
    this.delegate = delegate;
    this.method = method;
    try {
        this.requestUrl = new URL(requestUrl);
    }
    catch(Exception ex){
        ex.printStackTrace();
    }
}
@Override
public void onCompletion(OnCompleteListener delegate) {
    APIAccessTask(Context context, String requestUrl, String method,
List<Pair<String, String>> postData, OnCompleteListener delegate){
    this(context, requestUrl, method, delegate);
    this.postData = postData;
}
APIAccessTask(Context context, String requestUrl, String method,
List<Pair<String, String>> postData,
List<Pair<String, String>> headerData, OnCompleteListener delegate ) {
    this(context, requestUrl, method, postData, delegate);
    this.headerData = headerData;
}
@Override
protected void onPreExecute() {
    super.onPreExecute();
}
public class MultiPostUtility {
    @Override
    protected APIResponseObject doInBackground(String... params) {
        Log.d("debug", "url = " + requestUrl);
        try {
            urlConnection = (HttpURLConnection) requestUrl.openConnection();

            if(headerData != null) {
                for (Pair pair : headerData) {
                    urlConnection.setRequestProperty(pair.first.toString(), pair.
second.toString());
                }
            }

            urlConnection.setDoInput(true);
            urlConnection.setChunkedStreamingMode(0);
            urlConnection.setRequestMethod(method);
            urlConnection.connect();

            StringBuilder sb = new StringBuilder();

            if(!(method.equals("GET"))){
                OutputStream out = new BufferedOutputStream(urlConnection.
getOutputStream());
                BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(out, "UTF-8"));
                writer.write(getpostDataString(postData));
            }
        }
    }
}
```

```

        writer.flush();
        writer.close();
        out.close();
    }

    urlConnection.connect();
    responseCode = urlConnection.getResponseCode();
    if (responseCode == HttpURLConnection.HTTP_OK) {
        InputStream in = new BufferedInputStream(urlConnection.
getInputStream());
        BufferedReader reader = new BufferedReader(new
InputStreamReader(in, "UTF-8"));
        String line;

        while ((line = reader.readLine()) != null) {
            sb.append(line);
        }
    }

    return new APIResponseObject(responseCode, sb.toString());
}
catch(Exception ex){
    ex.printStackTrace();
}
return null;
}

@Override
protected void onPostExecute(APIResponseObject result) {
    delegate.onComplete(result);
    super.onPostExecute(result);
}

private String postDataString(List<Pair<String, String>> params) throws
UnsupportedEncodingException {
    StringBuilder result = new StringBuilder();
    boolean first = true;
    for(Pair<String, String> pair : params){
        if (first)
            first = false;
        else
            result.append("&");

        result.append(URLEncoder.encode(pair.first, "UTF-8"));
        result.append("=");
        result.append(URLEncoder.encode(pair.second, "UTF-8"));
    }
    return result.toString();
}
}

```

Использование

Используйте любой из приведенных конструкторов класса в зависимости от того, нужно ли передавать POST-данные или какие-либо дополнительные заголовки.

Метод `onComplete()` будет вызван после завершения выборки данных. Данные возвращаются в виде объекта класса `APIResponseObject`, который имеет код состояния, указывающий на HTTP-статус запроса, и строку, содержащую ответ. Этот ответ можно разобрать в своем классе, например, в XML или JSON.

Вызовите execute() на объекте класса для выполнения запроса, как показано в следующем примере:

```
class MainClass {
    String url = "https://example.com./api/v1/ex";
    String method = "POST";
    List<Pair<String, String>> postData = new ArrayList<>();

    postData.add(new Pair<>("email", "whatever"));
    postData.add(new Pair<>("password", "whatever"));

    new APIAccessTask(MainActivity.this, url, method, postData,
        new APIAccessTask.OnCompleteListener() {
            @Override
            public void onComplete(APIResponseObject result) {
                if (result.responseCode == HttpURLConnection.HTTP_OK) {
                    String str = result.response;
                    // Выполните здесь разбор XML/JSON
                }
            }
        }).execute();
}
```

74.6. Использование HttpURLConnection для multipart- и multiform-данных

Создание пользовательского класса для вызова запроса multipart- и multiform-данных HttpURLConnection:

MultipartUtility.java

```
public class MultipartUtility {
    private final String boundary;
    private static final String LINE_FEED = "\r\n";
    private HttpURLConnection httpConn;
    private String charset;
    private OutputStream outputStream;
    private PrintWriter writer;

    /**
     * Этот конструктор инициализирует новый HTTP POST-запрос с типом содержимого
     * установленным на multipart/form-данные
     *
     * @param requestURL
     * @param charset
     * @throws IOException
     */
    public MultipartUtility(String requestURL, String charset)
        throws IOException {
        this.charset = charset;

        // создается уникальная граница на основе временной метки
        boundary = "===" + System.currentTimeMillis() + "===";
        URL url = new URL(requestURL);
        httpConn = (HttpURLConnection) url.openConnection();
        httpConn.setUseCaches(false);
        httpConn.setDoOutput(true); // указывает на метод POST
        httpConn.setDoInput(true);
    }

    // методы для добавления полей в запрос
    public void addFormField(String name, String value) {
        // реализация метода
    }

    public void addFilePart(String fieldName, File file) {
        // реализация метода
    }

    public void addFilePart(String fieldName, File file, String fileName) {
        // реализация метода
    }

    public void addBinaryDataPart(String fieldName, byte[] data) {
        // реализация метода
    }

    public void addTextDataPart(String fieldName, String data) {
        // реализация метода
    }
}
```

```
    httpConn.setRequestProperty("Content-Type",
        "multipart/form-data; boundary=" + boundary);
    outputStream = httpConn.getOutputStream();
    writer = new PrintWriter(new OutputStreamWriter(outputStream, charset),
        true);
}

/**
 * Добавляет поле формы в запрос
 *
 * @param name имя поля
 * @param value значение поля
 */
public void addFormField(String name, String value) {
    writer.append("--" + boundary).append(LINE_FEED);
    writer.append("Content-Disposition: form-data; name=\"" + name + "\"");
    writer.append(LINE_FEED);
    writer.append("Content-Type: text/plain; charset=" + charset).append(
        LINE_FEED);
    writer.append(LINE_FEED);
    writer.append(value).append(LINE_FEED);
    writer.flush();
}

/**
 * Добавляет в запрос секцию загрузки файла
 *
 * @param fieldName атрибут имени в <input type="file" name="..." />
 * @param uploadFile файл для загрузки
 * @throws IOException
 */
public void addFilePart(String fieldName, File uploadFile)
    throws IOException {
    String fileName = uploadFile.getName();
    writer.append("--" + boundary).append(LINE_FEED);
    writer.append(
        "Content-Disposition: form-data; name=\"" + fieldName
        + "\"; filename=\"" + fileName + "\"")
        .append(LINE_FEED);
    writer.append(
        "Content-Type: "
        +URLConnection.guessContentTypeFromName(fileName))
        .append(LINE_FEED);
    writer.append("Content-Transfer-Encoding: binary").append(LINE_FEED);
    writer.append(LINE_FEED);
    writer.flush();

    FileInputStream inputStream = new FileInputStream(uploadFile);
    byte[] buffer = new byte[4096];
    int bytesRead = -1;
    while ((bytesRead = inputStream.read(buffer)) != -1) {
        outputStream.write(buffer, 0, bytesRead);
    }
    outputStream.flush();
    inputStream.close();
    writer.append(LINE_FEED);
    writer.flush();
}

/**
```

```

    * Добавляет в запрос поле заголовка.
    *
    * @param name - имя поля заголовка
    * @param value - значение поля заголовка
    */
    public void addHeaderField(String name, String value) {
        writer.append(name + ":" + value).append(LINE_FEED);
        writer.flush();
    }

    /**
     * Завершает запрос и получает ответ от сервера.
     *
     * @Возвращает список строк в качестве ответа в случае, если сервер вернул
     * статус OK, в противном случае выбрасывается исключение.
     * @throws IOException
     */
    public List<String> finish() throws IOException {
        List<String> response = new ArrayList<String>();
        writer.append(LINE_FEED).flush();
        writer.append("--" + boundary + "--").append(LINE_FEED);
        writer.close();

        // сначала проверяется код состояния сервера
        int status = httpConn.getResponseCode();
        if (status == HttpURLConnection.HTTP_OK) {
            BufferedReader reader = new BufferedReader(new InputStreamReader(
                httpConn.getInputStream()));
            String line = null;
            while ((line = reader.readLine()) != null) {
                response.add(line);
            }
            reader.close();
            httpConn.disconnect();
        } else {
            throw new IOException("Сервер вернул не-OK статус: " + status);
        }
        return response;
    }
}

```

Использование (Async-способ)

```
MultipartUtility multipart = new MultipartUtility(requestURL, charset);
```

```
// В вашем случае вы не добавляете данные формы, поэтому не обращайтесь на это
внимания
```

```
/* Для добавления значений параметров */
for (int i = 0; i < myFormDataArray.size(); i++) {
    multipart.addFormField(myFormDataArray.get(i).getParamName(),
        myFormDataArray.get(i).getParamValue());
}
```

```
// добавьте сюда свой файл.
```

```
/* Для добавления содержимого файла */
for (int i = 0; i < myFileArray.size(); i++) {
    multipart.addFilePart(myFileArray.getParamName(),
        new File(myFileArray.getFileName()));
}
```

```

List<String> response = multipart.finish();
Debug.e(TAG, "SERVER REPLIED:");
for (String line : response) {
    Debug.e(TAG, "Upload Files Response:::" + line);
// получите здесь ответ сервера.
    responseString = line;
}

```

74.7. Загрузка (POST) файла с помощью HttpURLConnection

Довольно часто возникает необходимость отправить или загрузить файл на удаленный сервер, например, изображение, видео, аудио или резервную копию базы данных приложения на удаленный частный сервер. Если предположить, что сервер ожидает POST-запрос с содержимым, то вот простой пример того, как выполнить эту задачу в Android.

Загрузка файлов осуществляется с помощью POST-запросов multipart/form-data. Это очень просто реализовать:

```

URL url = new URL(postTarget);
HttpURLConnection connection = (HttpURLConnection) url.openConnection();

String auth = "Bearer " + oauthToken; connection.
setRequestProperty("Authorization", basicAuth);

String boundary = UUID.randomUUID().toString(); connection.
setRequestMethod("POST"); connection.setDoOutput(true);
connection.setRequestProperty("Content-Type", "multipart/form-data;boundary=" +
boundary);
DataOutputStream request = new DataOutputStream(uc.getOutputStream());
request.writeBytes("--" + boundary + "\r\n");
request.writeBytes("Content-Disposition: form-data; name=\"description\"\r\n\r\n");
request.writeBytes(fileDescription + "\r\n");

request.writeBytes("--" + boundary + "\r\n");
request.writeBytes("Content-Disposition: form-data; name=\"file\"; filename=\"" +
file.fileName + "\"\r\n\r\n");
request.write(FileUtils.readFileToByteArray(file));
request.writeBytes("\r\n");

request.writeBytes("--" + boundary + "--\r\n");
request.flush();
int respCode = connection.getResponseCode();

switch(respCode) {
    case 200:
        // все прошло нормально - читаем ответ
        ...
        break;
    case 301:
    case 302:
    case 307:
        // обработать перенаправление - например, перепостить на новое место
        ...
        break;
    ...
default:
    // сделать что-то разумное
}

```

Разумеется, исключения необходимо будет перехватывать или объявлять как брошенные. Следует отметить несколько моментов, связанных с этим кодом.

1. `postTarget` – это целевой URL для POST; `oAuthToken` – токен аутентификации; `fileDescription` – описание файла, которое передается как значение поля `description`; `file` – отправляемый файл, имеет тип `java.io.File` – если у вас есть путь к файлу, то можно использовать `new File(filePath)`.

2. Устанавливает заголовок `Authorization` для аутентификации OAuth.

3. Использует Apache Common FileUtils для чтения файла в байтовый массив – если содержимое файла уже находится в байтовом массиве или каким-либо другим способом в памяти, то читать его не нужно.

Глава 75. URL обратного вызова

75.1. Пример URL-адреса обратного вызова с использованием Instagram OAuth

Одним из примеров использования *URL обратного вызова* является OAuth. Рассмотрим это на примере входа в Instagram: если пользователь вводит свои учетные данные и нажимает кнопку “Войти”, Instagram проверяет их и возвращает `access_token`. Нам нужен этот `access_token` в нашем приложении.

Для того чтобы наше приложение могло прослушивать такие ссылки, необходимо добавить URL обратного вызова в нашу активность. Это можно сделать, добавив в активность фильтр `<intent-filter>`, который будет реагировать на URL обратного вызова. Предположим, что наш URL обратного вызова – это `appSchema://appName.com`. Тогда в файл `Manifest.xml` необходимо добавить следующие строки для нужной активности:

```
<action android:name="android.intent.action.VIEW" />
<category android:name="android.intent.category.BROWSABLE" />
<data android:host="appName.com" android:scheme="appSchema" />
```

Пояснение к приведенным выше строкам:

- `<category android:name="android.intent.category.BROWSABLE" />` позволяет целевой активности запускать веб-браузер для отображения данных, на которые ссылается ссылка.
- `<data android:host="appName.com" android:scheme="appSchema" />` задает нашу схему и хост нашего URL обратного вызова.
- Все вместе эти строки приведут к тому, что при вызове URL обратного вызова в браузере будет открываться определенная активность.

Теперь, чтобы получить содержимое URL в активности, необходимо переопределить метод `onResume()` следующим образом:

```
@Override
public void onResume() {
    // Следующая строка вернет "appSchema://appName.com".
    String CALLBACK_URL = getResources().getString(R.string.insta_callback);
    Uri uri = getIntent().getData();
    if (uri != null && uri.toString().startsWith(CALLBACK_URL)) {
        String access_token = uri.getQueryParameter("access_token");
    }
    // Здесь выполняются другие операции.
}
```

Теперь вы получили от Instagram `access_token`, который используется в различных конечных точках API Instagram.

Глава 76. Всплывающее уведомление (Snackbar)

Параметр	Описание
view	View: представление, из которого будет найдено родительское.
text	CharSequence: Текст для отображения. Может быть форматированным текстом.
resId	int: Идентификатор используемого строкового ресурса. Может быть форматированным текстом.
duration	int: Продолжительность отображения сообщения. Может быть LENGTH_SHORT, LENGTH_LONG или LENGTH_INDEFINITE.

76.1. Создание простого всплывающего уведомления

Может быть выполнено следующим образом:

```
Snackbar.make(view, "Текст для отображения", Snackbar.LENGTH_LONG).show();
```

Представление (view) используется для поиска подходящего родителя, который можно использовать для отображения панели Snackbar. Обычно это CoordinatorLayout, определенный в XML, который позволяет добавить такие функции, как пролистывание для удаления и автоматическое перемещение других виджетов (например, FloatingActionButton). Если CoordinatorLayout отсутствует, то используется представление содержимого декора окна.

Очень часто мы также добавляем действие на Snackbar. Частым примером является действие “Отменить”.

```
Snackbar.make(view, "Текст для отображения", Snackbar.LENGTH_LONG)
    .setAction("UNDO", new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            // поместите сюда свою логику для выполнения
        }
    })
    .show();
```

Вы можете создать Snackbar и показать его позже:

```
Snackbar snackbar = Snackbar.make(view, "Текст для отображения", Snackbar.LENGTH_LONG);
snackbar.show();
```

Если вы хотите изменить цвет текста у Snackbar:

```
Snackbar snackbar = Snackbar.make(view, "Текст для отображения", Snackbar.LENGTH_LONG);
View view = snackbar.getView();
TextView textView = (TextView) view.findViewById(android.support.design.R.id.
snackbar_text);
textView.setTextColor(Color.parseColor("#FF4500"));
snackbar.show();
```

По умолчанию Snackbar отключается при проведении пальцем вправо. В данном примере показано, как отключить его при проведении пальцем влево: <http://stackoverflow.com/a/41790613/3732887>.

76.2. Пользовательское всплывающее уведомление

Функция настройки Snackbar:

```
public static Snackbar makeText(Context context, String message, int duration) {
    Activity activity = (Activity) context;
```

```

View layout;
Snackbar snackbar = Snackbar
    .make(activity.findViewById(android.R.id.content), message, duration);
layout = snackbar.getView();
//установка цвета фона
layout.setBackgroundColor(context.getResources().getColor(R.color.
orange)); android.widget.TextView text = (android.widget.TextView) layout.
findViewById(android.support.design.R.id.snackbar_text);
//установка цвета шрифта
text.setTextColor(context.getResources().getColor(R.color.white));
Typeface font = null;
//Установка шрифта
font = Typeface.createFromAsset(context.getAssets(),
"DroidSansFallbackanmol256.ttf");
text.setTypeface(font);
return snackbar;
}

```

Пример вызова этой функции из фрагмента или активности:

```
SnackBar.makeText(MyActivity.this, "Please Locate your address at Map", Snackbar.LENGTH_SHORT).show();
```

76.3. Пользовательское всплывающее уведомление (View не требуется)

Рассмотрим случай создания Snackbar без необходимости передачи в него представления View, весь макет создается в android.R.id.content.

```

public class CustomSnackBar {

    public static final int STATE_ERROR = 0;
    public static final int STATE_WARNING = 1;
    public static final int STATE_SUCCESS = 2;
    public static final int VIEW_PARENT = android.R.id.content;

    public CustomSnackBar(View view, String message, int actionType) {
        super();

        Snackbar snackbar = Snackbar.make(view, message, Snackbar.LENGTH_LONG);
        View sbView = snackbar.getView();
        TextView textView = (TextView) sbView.findViewById(android.support.
design.R.id.snackbar_text);
        textView.setTextColor(Color.parseColor("#ffffff"));
        textView.setTextSize(TypedValue.COMPLEX_UNIT_SP, 14);
        textView.setGravity(View.TEXT_ALIGNMENT_CENTER);
        textView.setLayoutDirection(View.LAYOUT_DIRECTION_RTL);

        switch (actionType) {
            case STATE_ERROR:
                snackbar.getView().setBackgroundColor(Color.parseColor("#F12B2B"));
                break;
            case STATE_WARNING:
                snackbar.getView().setBackgroundColor(Color.parseColor("#000000"));
                break;
            case STATE_SUCCESS:
                snackbar.getView().setBackgroundColor(Color.parseColor("#7ED321"));
                break;
        }
    }
}

```

```

        snackbar.show();
    }
}

```

Для класса вызова:

```

new CustomSnackBar(findViewById(CustomSnackBar.VIEW_PARENT), "message",
CustomSnackBar.STATE_ERROR);

```

76.4. Всплывающее уведомление с обратным вызовом

Вы можете использовать Snackbar.Callback для получения информации о том, что Snackbar-панель была отменена пользователем или закончилось время:

```

Snackbar.make(getView(), "Hi Snackbar!", Snackbar.LENGTH_LONG).setCallback( new
Snackbar.Callback() {
    @Override
    public void onDismissed(Snackbar snackbar, int event) {
        switch(event) {
            case Snackbar.Callback.DISMISS_EVENT_ACTION:
                Toast.makeText(getApplicationContext(), "Нажал на действие",
Toast.LENGTH_LONG).show();
                break;
            case Snackbar.Callback.DISMISS_EVENT_TIMEOUT:
                Toast.makeText(getApplicationContext(), "Time out", Toast.LENGTH_LONG).show();
                break;
        }
    }
    @Override
    public void onShown(Snackbar snackbar) {
        Toast.makeText(getApplicationContext(), "Это мой надоедливый сводный брат", Toast.
LENGTH_LONG).show();
    }
}).setAction("Go!", new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        }
}).show();

```

76.5. Snackbar и Toast: что лучше использовать?

Тосты обычно используются в тех случаях, когда мы хотим вывести на экран информацию о некотором действии, которое успешно (или нет) произошло, и это действие не требует от пользователя каких-либо других действий. Например, когда сообщение отправлено:

```
Toast.makeText(this, "Сообщение отправлено!", Toast.LENGTH_SHORT).show();
```

Snackbar также используется для отображения информации. Но на этот раз мы можем предоставить пользователю возможность совершить какое-либо действие. Например, пользователь по ошибке удалил фотографию и хочет ее вернуть. Мы можем предоставить ему Snackbar с действием "Undo". Например:

```

Snackbar.make(getApplicationContext(), "Изображение удалено", Snackbar.LENGTH_SHORT)
.setAction("Undo", new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Восстановить изображение
    }
});

```

```

        }
    })
    .show();
}

```

Выводы: Toasts используются в тех случаях, когда нам не нужно взаимодействие с пользователем. Snackbars используются для того, чтобы позволить пользователю совершить другое действие или отменить предыдущее.

76.6. Пользовательское всплывающее уведомление со значком Undo

В данном примере показана белая панель Snackbar с пользовательским значком Undo:

```

Snackbar customBar = Snackbar.make(view, "Текст для отображения", Snackbar.LENGTH_
LONG);
customBar.setAction("UNDO", new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        //Поместите сюда логику для кнопки отмены
    }
});
View sbView = customBar.getView();
//Изменение фона на белый
sbView.setBackgroundColor(Color.WHITE);

TextView snackText = (TextView) sbView.findViewById(android.support.design.R.id.
snackbar_text);
if (snackText!=null) {
    //Изменение цвета текста на Black
    snackText.setTextColor(Color.BLACK);
}

TextView actionText = (TextView) sbView.findViewById(android.support.design.R.id.
snackbar_action);
if (actionText!=null) {
    // Установка пользовательского значка Undo
    actionText.setCompoundDrawablesRelativeWithIntrinsicBounds(R.drawable.custom_
    undo, 0, 0, 0);
}
customBar.show();

```

Глава 77. Виджеты

77.1. Объявление в манифесте

Объявите класс AppWidgetProvider в файле AndroidManifest.xml вашего приложения. Например:

```

<receiver android:name="ExampleAppWidgetProvider" >
<intent-filter>
    <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
</intent-filter>
<meta-data android:name="android.appwidget.provider"
    android:resource="@xml/example_appwidget_info" />
</receiver>

```

77.2. Метаданные

Добавьте метаданные AppWidgetProviderInfo в res/xml:

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="40dp"
    android:minHeight="40dp"
    android:updatePeriodMillis="86400000"
    android:previewImage="@drawable/preview"
    android:initialLayout="@layout/example_appwidget"
    android:configure="com.example.android.ExampleAppWidgetConfigure"
    android:resizeMode="horizontal|vertical"
    android:widgetCategory="home_screen">
</appwidget-provider>
```

77.3. Класс AppWidgetProvider

Наиболее важным обратным вызовом AppWidgetProvider является `onUpdate()`. Он вызывается каждый раз при добавлении виджета.

```
public class ExampleAppWidgetProvider extends AppWidgetProvider {

    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
        final int N = appWidgetIds.length;

        // Выполните эту процедуру цикла для каждого виджета App Widget, принадлежащего данному провайдеру
        for (int i=0; i<N; i++) {
            int appWidgetId = appWidgetIds[i];

            // Создание намерения для запуска ExampleActivity
            Intent intent = new Intent(context, ExampleActivity.class);
            PendingIntent pendingIntent = PendingIntent.getActivity(context, 0,
                intent, 0);

            // Получение макета для виджета App Widget и прикрепление слушателя клика к кнопке
            RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.
                appwidget_provider_layout);
            views.setOnClickPendingIntent(R.id.button, pendingIntent);

            // Сообщим AppWidgetManager о необходимости выполнить обновление текущего виджета приложения
            appWidgetManager.updateAppWidget(appWidgetId, views);
        }
    }
}
```

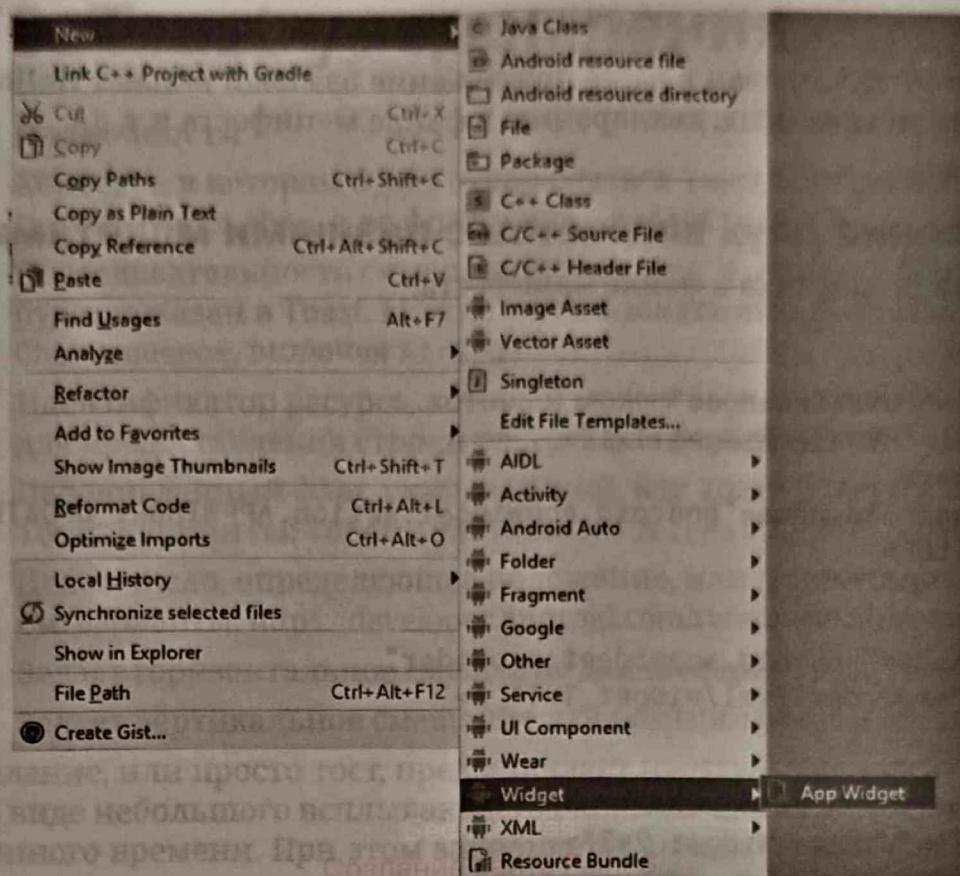
`onAppWidgetOptionsChanged()` вызывается при размещении или изменении размера виджета.

`onDeleted(Context, int[])` вызывается при удалении виджета.

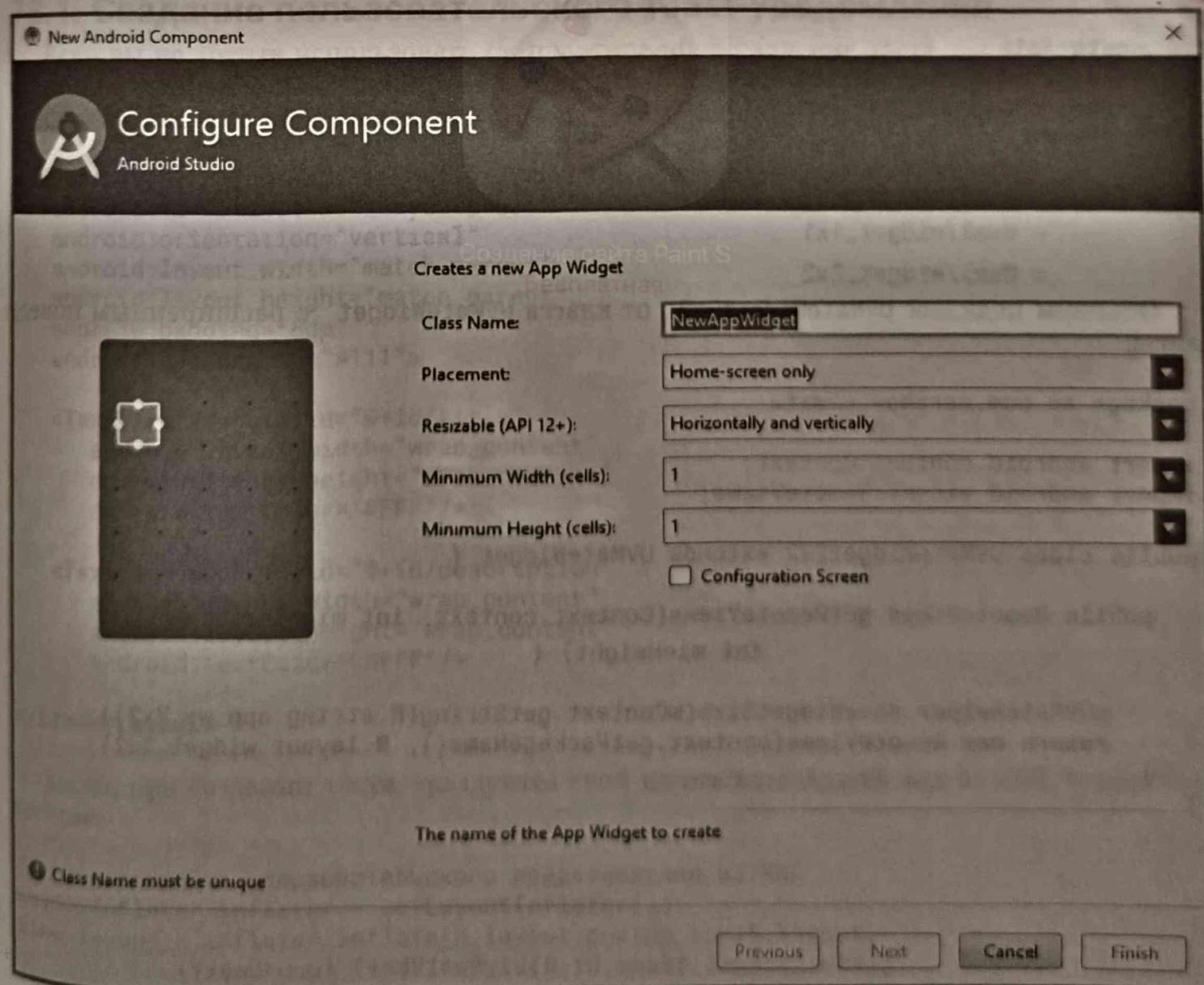
77.4. Создание и встраивание базового виджета

Android Studio позволяет создать и интегрировать базовый виджет в ваше приложение за 2 шага.

Выполните `New > Widget > App Widget`.



На экране появится окно, как показано ниже; заполните в нем поля:



Готово.

Будет создан и интегрирован в ваше приложение базовый виджет **HelloWorld** (включая файл макета, файл метаданных, декларацию в файле манифеста и т. д.).

77.5. Объявление двух виджетов с разными макетами

1. Объявим два приемника в файле манифеста:

```
<receiver
    android:name=".UVMateWidget"
    android:label="UVMate Widget 1x1">
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>

    <meta-data
        android:name="android.appwidget.provider"
        android:resource="@xml/widget_1x1" />
</receiver>
<приёмник
    android:name=".UVMateWidget2x2"
    android:label="UVMate Widget 2x2">
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>

    <meta-data
        android:name="android.appwidget.provider"
        android:resource="@xml/widget_2x2" />
</receiver>
```

2. Создадим два макета:

- @xml/widget_1x1
- @xml/widget_2x2

Объявим подкласс **UVMateWidget2x2** от класса **UVMateWidget** с расширенным поведением:

```
package au.com.aershov.uvmate;

import android.content.Context;
import android.widget.RemoteViews;

public class UVMateWidget2x2 extends UVMateWidget {

    public RemoteViews getRemoteViews(Context context, int minWidth,
                                      int minHeight) {

        mUVMateHelper.saveWidgetSize(mContext.getString(R.string.app_ws_2x2));
        return new RemoteViews(context.getPackageName(), R.layout.widget_2x2);
    }
}
```

Глава 78. Toast-уведомления

Параметр	Подробности
context	Контекст, в котором будет отображаться Toast. В активности обычно используется <code>this</code> , а во фрагменте – <code>getActivity()</code>
text	Последовательность символов <code>CharSequence</code> , определяющая, какой текст будет показан в Toast. Можно использовать любой объект, реализующий <code>CharSequence</code> , включая <code>String</code>
resId	Идентификатор ресурса, который может быть использован для предоставления строки ресурса для отображения в уведомлении
duration	Целочисленный флаг, указывающий, как долго будет отображаться Toast. Варианты: <code>Toast.LENGTH_SHORT</code> и <code>Toast.LENGTH_LONG</code>
gravity	Целое число, определяющее положение, или “гравитацию” Toast. См. варианты: https://developer.android.com/reference/android/view/Gravity.html
xOffset	Задает горизонтальное смещение для позиции Toast-уведомления
yOffset	Задает вертикальное смещение для позиции уведомления
	Toast-уведомление, или просто тост, предоставляет простую обратную связь о выполняемой операции в виде небольшого всплывающего окна; оно автоматически исчезает по истечении определенного времени. При этом заполняется только необходимый для сообщения объем пространства, а текущая активность остается видимой и интерактивной.

78.1. Создание пользовательского Toast-уведомления

Если вы не хотите использовать стандартное представление тоста, вы можете создать свое собственное с помощью метода `setView(View)` на элементе `Toast`.

Сначала создайте XML-макет, который вы хотите использовать для уведомления:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/toast_layout_root"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="8dp"
    android:background="#111">

    <TextView android:id="@+id/title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#FFF"/>

    <TextView android:id="@+id/description"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#FFF"/>

</LinearLayout>
```

Затем, при создании тоста, «раздуйте» свой пользовательский вид из XML и вызовите `setView`:

```
// Разворачивание пользовательского представления из XML
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.custom_toast_layout,
    (ViewGroup) findViewById(R.id.toast_layout_root));
```

```
// Устанавливаем заголовок и описание TextView из нашего пользовательского макета
TextView title = (TextView) layout.findViewById(R.id.title);
title.setText("Заголовок тоста");
TextView description = (TextView) layout.findViewById(R.id.description);
description.setText("Описание тоста");

// Создание и отображение объекта Toast
Toast toast = new Toast(getApplicationContext());
toast.setGravity(Gravity.CENTER, 0, 0);
toast.setDuration(Toast.LENGTH_LONG);
toast.setView(layout);
toast.show();
```

78.2. Установка положения Toast-уведомления

Стандартный тост появляется в нижней части экрана, выровненный по центру по горизонтали. Изменить это положение можно с помощью команды `setGravity(int, int, int)`. Эта функция принимает три параметра: константу Gravity, смещение x-позиции и смещение y-позиции.

Например, если вы решили, что тост должен отображаться в левом верхнем углу, вы можете задать гравитацию следующим образом:

```
toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);
```

78.3. Отображение Toast-уведомления

Чтобы вывести на экран простое Toast-уведомление, мы можем сделать следующее:

```
// Объявление параметров, которые будут использоваться для тоста

Context context = getApplicationContext();
// в Activity можно также использовать "this"
// во фрагменте можно использовать getActivity()

CharSequence message = "Я тост Android!";
int duration = Toast.LENGTH_LONG; // Toast.LENGTH_SHORT - другой вариант

// Создайте объект Toast и покажите его!
Toast myToast = Toast.makeText(context, message, duration);
myToast.show();
```

...или, чтобы показать тост в строке, не удерживая объект Toast, вы можете использовать:

```
Toast.makeText(context, "Дзинь! Ваш тост готов.", Toast.LENGTH_SHORT).show();
```

Важно: убедитесь, что метод `show()` вызывается из потока пользовательского интерфейса. Если вы пытаетесь показать тост из другого потока, то можно, например, использовать метод `runOnUiThread` в активности.

Если этого не сделать, то есть попытаться изменить пользовательский интерфейс путем создания тоста, будет выброшен `RuntimeException`, который будет выглядеть следующим образом:

```
java.lang.RuntimeException: Can't create handler inside thread that has not called
Looper.prepare() (Невозможно создать обработчик внутри потока, который не вызвал
Looper.prepare())
```

Простейший способ обработки этого исключения – использование функции `runOnUiThread`: синтаксис показан ниже.

```
runOnUiThread(new Runnable() {
    @Override
    public void run() {
        // Ваш код здесь
    }
});
```

78.4. Отображение Toast-уведомления над экранной клавиатурой

По умолчанию Android отображает тост-сообщения в нижней части экрана, даже если отображается клавиатура. Так Toast будет отображаться над клавиатурой:

```
public void showMessage(final String message, final int length) {
    View root = findViewById(android.R.id.content);
    Toast toast = Toast.makeText(this, message, length);
    int yOffset = Math.max(0, root.getHeight() - toast.getYOffset());
    toast.setGravity(Gravity.TOP | Gravity.CENTER_HORIZONTAL, 0, yOffset);
    toast.show();
}
```

78.5. Потокобезопасный способ отображения Toast-уведомления (для всего приложения)

```
public class MainApplication extends Application {

    private static Context context; //контекст приложения

    private Handler mainThreadHandler;
    private Toast toast;

    public Handler getMainThreadHandler() {
        if (mainThreadHandler == null) {
            mainThreadHandler = new Handler(Looper.getMainLooper());
        }
        return mainThreadHandler;
    }

    @Override public void onCreate() {
        super.onCreate();
        context = this;
    }

    public static MainApplication getApp(){
        return (MainApplication) context;
    }

    /**
     * Потокобезопасный способ отображения тостов.
     * @param message
     * @param duration
     */
    public void showToast(final String message, final int duration) {
        getMainThreadHandler().post(new Runnable() {
            @Override
```

```

public void run() {
    if (!TextUtils.isEmpty(message)) {
        if (toast != null) {
            toast.cancel(); //удаление текущего тоста, если он виден
            toast.setText(message);
        } else {
            toast = Toast.makeText(App.this, message, duration);
        }
        toast.show();
    }
}
);
}

```

Не забудьте добавить MainApplication в manifest.

Теперь вызовите его из любого потока, чтобы вывести тост-уведомление.

```
MainApplication.getApp().showToast("Ваше сообщение", Toast.LENGTH_LONG);
```

78.6. Потокобезопасный способ отображения Toast-уведомления (для AsyncTask)

Если вы хотите сохранять потокобезопасность тостов и не хотите расширять приложение, убедитесь, что они отображаются в секции PostExecute ваших задач AsyncTasks.

```

public class MyAsyncTask extends AsyncTask <Void, Void, Void> {

    @Override
    protected Void doInBackground(Void... params) {
        // Фоновые задачи выполнить здесь
    }

    @Override
    protected void onPostExecute(Void aVoid) {
        // Здесь отображаются сообщения о тостах
        Toast.makeText(context, "Динь! Ваш тост готов.", Toast.LENGTH_SHORT).show();
    }
}

```

Глава 79. Создание синглтон-класса для тост-уведомления

Тостовые уведомления являются наиболее простым способом предоставления обратной связи пользователю. Если нам необходимо создать более настраиваемое и многократно используемое тост-уведомление, мы можем реализовать его самостоятельно с помощью пользовательского макета. Более того, использование паттерна проектирования Singleton упростит поддержку и развитие класса пользовательских тостов.

Если в вашем приложении необходимо показывать сообщения об успехе, предупреждении и опасности для различных случаев использования, вы можете использовать этот класс, модифицировав его в соответствии с вашими требованиями.

```
public class ToastGenerate {
    private static ToastGenerate ourInstance;

    public ToastGenerate (Context context) {
        this.context = context;
    }

    public static ToastGenerate getInstance(Context context) {
        if (ourInstance == null)
            ourInstance = new ToastGenerate(context);
        return ourInstance;
    }

    // передать сообщение и тип сообщения в этот метод
    public void createToastMessage(String message,int type){

        // наполнение пользовательского макета
        LayoutInflater layoutInflater = (LayoutInflater) context.
        getSystemService(Context.LAYOUT_INFLATER_SERVICE);

        LinearLayout toastLayout = (LinearLayout) layoutInflater.inflate(R.
        layout.layout_custo_me_toast,null);
        TextView toastShowMessage = (TextView) toastLayout.findViewById(R.
        id.textCustomToastTopic);

        switch (type){
            case 0:
                //если тип сообщения равен 0, будет вызван метод fail toaster
                createFailToast(toastLayout,toastShowMessage,message);
                break;
            case 1:
                //если тип сообщения равен 1, будет вызван метод success toaster
                createSuccessToast(toastLayout,toastShowMessage,message);
                break;
            case 2:
                createWarningToast( toastLayout, toastShowMessage, message);
                //если тип сообщения = 2, то будет вызван метод warning toaster
                break;
            default:
                createFailToast(toastLayout,toastShowMessage,message);
        }
    }

    //метод неуспешного тост-сообщения FailToast
    private final void createFailToast(LinearLayout toastLayout, TextView
    toastMessage, String message){
        toastLayout.setBackgroundColor(context.getResources().getColor(R.color.
        button_alert_normal));
        toastMessage.setText(message);
        toastMessage.setTextColor(context.getResources().getColor(R.color.
        white));
        showToast(context,toastLayout);
    }

    //метод предупреждения warning toast message method
    private final void createWarningToast( LinearLayout toastLayout, TextView
    toastMessage, String message) {
        toastLayout.setBackgroundColor(context.getResources().getColor(R.color.
        warning_toast));
    }
}
```

```

        toastMessage.setText(message);
        toastMessage.setTextColor(context.getResources().getColor(R.color.
white));
        showToast(context, toastLayout);
    }
    //метод успешного тост-сообщения success toast message method
    private final void createSuccessToast(LinearLayout toastLayout, TextView
toastMessage, String message){
        toastLayout.setBackgroundColor(context.getResources().getColor(R.color.
success_toast));

        toastMessage.setText(message);
        toastMessage.setTextColor(context.getResources().getColor(R.color.
white));
        showToast(context, toastLayout);
    }
    private void showToast(View view){
        Toast toast = new Toast(context);
        toast.setGravity(Gravity.TOP, 0, 0); // показать сообщение в верхней
части устройства
        toast.setDuration(Toast.LENGTH_SHORT);
        toast.setView(view);
        toast.show();
    }
}

```

Глава 80. Интерфейсы

80.1. Пользовательский слушатель

Определение интерфейса

//В этом интерфейсе можно определить сообщения, которые будут отправляться
владельцу.

```

public interface MyCustomListener {
    //В данном случае мы имеем два сообщения,
    //первое, которое отправляется при успешном завершении процесса.
    void onSuccess(List<Bitmap> bitmapList);
    //Второе – если процесс завершится неудачей.
    void onFailure(String error);
}

```

Создание слушателя

На следующем этапе нам необходимо определить в объекте переменную экземпляра, которая будет отправлять обратный вызов через MyCustomListener. И добавить сеттер для нашего слушателя.

```

public class SampleClassB {
    private MyCustomListener listener;

    public void setMyCustomListener(MyCustomListener listener) {
        this.listener = listener;
    }
}

```



Реализация слушателя

Теперь в другом классе мы можем создать экземпляр SampleClassB.

```
public class SomeActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        SampleClassB sampleClass = new SampleClassB();
    }
}
```

Далее мы можем установить наш слушатель на sampleClass двумя способами.

1. Реализовав MyCustomListener в нашем классе:

```
public class SomeActivity extends Activity implements MyCustomListener {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        SampleClassB sampleClass = new SampleClassB();
        sampleClass.setMyCustomListener(this);
    }

    @Override
    public void onSuccess(List<Bitmap> bitmapList) {
    }

    @Override
    public void onFailure(String error) {
    }
}
```

2. Просто инстанцировать анонимный внутренний класс:

```
public class SomeActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        SampleClassB sampleClass = new SampleClassB();
        sampleClass.setMyCustomListener(new MyCustomListener() {
            @Override
            public void onSuccess(List<Bitmap> bitmapList) {
            }

            @Override
            public void onFailure(String error) {
            }
        });
    }
}
```

Слушатель триггера

```
public class SampleClassB {
    private MyCustomListener listener;

    public void setMyCustomListener(MyCustomListener listener) {
```

```

        this.listener = listener;
    }

    public void doSomething() {
        fetchImages();
    }

    private void fetchImages() {
        AsyncImagefetch imageFetch = new AsyncImageFetch();
        imageFetch.start(new Response<Bitmap>() {
            @Override
            public void onDone(List<Bitmap> bitmapList, Exception e) {
                //при необходимости выполнить некоторые действия

                //Проверяем, установлен ли слушатель.
                if(listener == null)
                    return;
                // Выполнить соответствующее событие. bitmapList или сообщение
                // об ошибке будут отправлены в класс, устанавливающий слушателя
                if(e == null)
                    listener.onSuccess(bitmapList);
                else
                    listener.onFailure(e.getMessage());
            }
        });
    }
}

```

80.2. Базовый слушатель

Использование шаблонов «слушатель» (listener) или «наблюдатель» (observer) является наиболее распространенной стратегией создания асинхронных обратных вызовов в Android-разработке.

```

public class MyCustomObject {

    //1 - Определение интерфейса
    public interface MyCustomObjectListener {
        public void onAction(String action);
    }

    //2 - Объявление объекта слушателя
    private MyCustomObjectListener listener;

    // и инициализация его в конструкторе
    public MyCustomObject() {
        this.listener = null;
    }

    //3 - Создание сеттера слушателя
    public void setCustomObjectListener(MyCustomObjectListener listener) {
        this.listener = listener;
    }

    // 4 - Запуск события слушателя
    public void makeSomething(){
        if (this.listener != null){
            listener.onAction("hello!");
        }
    }
}

```

В вашей активности:

```
public class MyActivity extends Activity {
    public final String TAG = "MyActivity";

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

        MyCustomObject mObj = new MyCustomObject();

        //5 - Реализовать обратный вызов слушателя
        mObj.setCustomObjectListener(new MyCustomObjectListener() {
            @Override
            public void onAction(String action) {
                Log.d(TAG, "Значение: "+action);
            }
        });
    }
}
```

Глава 81. Аниматоры

81.1. Анимация TransitionDrawable

В данном примере отображается транзакция для представления изображений, содержащая только два изображения (можно использовать и большее количество изображений – одно за другим для позиций первого и второго слоя после каждой транзакции в виде цикла).

- добавим массив изображений в `res/values/arrays.xml`:

```
<resources>
    <array
        name="splash_images">
        <item>@drawable/splash_image_first</item>
        <item>@drawable/splash_image_second</item>
    </array>
</resources>

private Drawable[] backgroundsDrawableArrayForTransition;
private TransitionDrawable transitionDrawable;

private void backgroundAnimTransAction() {
    // установим массив изображений
    Resources resources = getResources();
    TypedArray icons = resources.obtainTypedArray(R.array.splash_images);

    @SuppressWarnings(" ResourceType")
    Drawable drawable = icons.getDrawable(0); // завершающее изображение
```

```

@SuppressLint("ResourceType")
Drawable drawableTwo = icons.getDrawable(1); // начальное изображение

backgroundsDrawableArrayForTransition = new Drawable[2];
backgroundsDrawableArrayForTransition[0] = drawable;
backgroundsDrawableArrayForTransition[1] = drawableTwo;
transitionDrawable = new
TransitionDrawable(backgroundsDrawableArrayForTransition);

// здесь ваше представление изображения - backgroundImageView
backgroundImageView.setImageDrawable(transitionDrawable);
transitionDrawable.startTransition(4000);

transitionDrawable.setCrossFadeEnabled(false); // вызов открытых методов
}

```

81.2. Анимация с постепенным появлением и затуханием

Для того чтобы заставить вид view медленно появиться (fade in) или исчезнуть (fade out), используйте ObjectAnimator. Как показано в приведенном ниже коде, задайте длительность с помощью .setDuration(millis), где параметр millis – это длительность (в миллисекундах) анимации. В приведенном ниже коде вид будет появляться и исчезать в течение 500 миллисекунд, или 1/2 секунды. Чтобы запустить анимацию ObjectAnimator, вызовите .start(). После завершения анимации вызывается onAnimationEnd(Animator animation). Здесь следует установить видимость представления на View.GONE или View.VISIBLE.

```

import android.animation.Animator;
import android.animation.AnimatorListenerAdapter;
import android.animation.ValueAnimator;

void fadeOutAnimation(View viewToFadeOut) {
    ObjectAnimator fadeOut = ObjectAnimator.ofFloat(viewToFadeOut, "alpha", 1f, 0f);
    fadeOut.setDuration(500);
    fadeOut.addListener(new AnimatorListenerAdapter() {
        @Override
        public void onAnimationEnd(Animator animation) {
            // Мы хотим установить для вида значение GONE после его затухания, чтобы он
            // действительно исчез из макета и не занимал места.
            viewToFadeOut.setVisibility(View.GONE);
        }
    });
    fadeOut.start();
}

void fadeInAnimation(View viewToFadeIn) {
    ObjectAnimator fadeIn = ObjectAnimator.ofFloat(viewToFadeIn, "alpha", 0f, 1f);
    fadeIn.setDuration(500);

    fadeIn.addListener(new AnimatorListenerAdapter() {
        @Override
        public void onAnimationStart(Animator animation) {
            // Мы хотим установить вид на VISIBLE, но со значением альфа 0, чтобы он
            // отображался невидимым в макете.
            viewToFadeIn.setVisibility(View.VISIBLE);
            viewToFadeIn.setAlpha(0);
        }
    });
}

```

```

});
```

} // появление

```

fadeIn.start();
}
```

81.3. ValueAnimator

ValueAnimator представляет собой простой способ анимирования значения (определенного типа, например, int, float и т.д.). Обычно он используется следующим образом:

1. Создайте ValueAnimator, который будет анимировать значение от min до max.
2. Добавьте UpdateListener, в котором вы будете использовать вычисленное значение анимации (которое можно получить с помощью getAnimatedValue()).

Существует два способа создания ValueAnimator:

(пример кода анимирует float от 20f до 40f за 250 мс)

1. Из xml (поместите его в папку /res/ animator/):

```
<animator xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="250"
    android:valueFrom="20"
    android:valueTo="40"
    android:valueType="floatType"/>
```

```
ValueAnimator animator = (ValueAnimator) AnimatorInflater.loadAnimator(context,
    R.animator.example_animator);
animator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
    @Override
    public void onAnimationUpdate(ValueAnimator anim) {
        // ... использовать anim.getAnimatedValue()
    }
});
// установите все остальное, связанное с анимацией (интерpolator и т.д.)
animator.start();
```

2. Из кода:

```
ValueAnimator animator = ValueAnimator.ofFloat(20f, 40f);
animator.setDuration(250);
animator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
    @Override
    public void onAnimationUpdate(ValueAnimator anim) {
        // использовать anim.getAnimatedValue()
    }
});
// установите все остальное, связанное с анимацией (интерpolator и т.д.)
animator.start();
```

81.4. Разворачивание и сворачивание анимации представления

```
public class ViewAnimationUtils {
```

```

    public static void expand(final View v) {
        v.measureLayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENT);
        final int targetHeight = v.getMeasuredHeight();

        v.getLayoutParams().height = 0;
        v.setVisibility(View.VISIBLE);
        Animation a = new Animation()
        {

            @Override
            protected void applyTransformation(float interpolatedTime,
```

```

        Transformation t) {
            v.getLayoutParams().height = interpolatedTime == 1
                ? LayoutParams.WRAP_CONTENT
                : (int)(targetHeight * interpolatedTime);
            v.requestLayout();
        }

        @Override
        public boolean willChangeBounds() {
            return true;
        }
    };

    a.setDuration((int)(targetHeight / v.getContext().getResources()
        .getDisplayMetrics().density));
    v.startAnimation(a);
}

public static void collapse(final View v) {
    final int initialHeight = v.getMeasuredHeight();

    Animation a = new Animation()
    {
        @Override
        protected void applyTransformation(float interpolatedTime,
            Transformation t) {
            if(interpolatedTime == 1){
                v.setVisibility(View.GONE);
            }else{
                v.getLayoutParams().height = initialHeight - (int)
                (initialHeight * interpolatedTime);

                v.requestLayout();
            }
        }

        @Override
        public boolean willChangeBounds() {
            return true;
        }
    };

    a.setDuration((int)(initialHeight / v.getContext().getResources()
        .getDisplayMetrics().density));
    v.startAnimation(a);
}
}

```

81.5. ObjectAnimator

ObjectAnimator является подклассом ValueAnimator с добавленной возможностью установки вычисляемого значения в свойство целевого представления View.

Как и в случае с ValueAnimator, у ObjectAnimator есть два способа создания:

(пример кода анимирует альфа-канал представления с 0.4f до 0.2f за 250 мс)

1. Из xml (поместите его в папку/res/Animator)

```
<objectAnimator xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="250"
```

```

        android:propertyName="alpha"
        android:valueFrom="0.4"
        android:valueTo="0.2"
        android:valueType="floatType"/>
ObjectAnimator animator = (ObjectAnimator) AnimatorInflater.loadAnimator(context,
R.animator.example_animator);
animator.setTarget(exampleView);
// установить все необходимое, связанное с анимацией (интерполятор и т.д.)
animator.start();

```

2. Из кода:

```

ObjectAnimator animator = ObjectAnimator.ofFloat(exampleView, View.ALPHA, 0.4f, 0.2f);
animator.setDuration(250);
// установить все необходимое, связанное с анимацией (интерполятор и т.д.)
animator.start();

```

81.6. ViewPropertyAnimator

ViewPropertyAnimator – это упрощенный и оптимизированный способ анимации свойств представления **View**.

Каждое **View** имеет объект **ViewPropertyAnimator**, доступный через метод **animate()**. С его помощью можно простым вызовом анимировать сразу несколько свойств. Каждый метод **ViewPropertyAnimator** задает **целевое** значение определенного параметра, к которому должен анимироваться **ViewPropertyAnimator**.

```

View exampleView = ...;
exampleView.animate()
    .alpha(0.6f)
    .translationY(200)
    .translationXBy(10)
    .scaleX(1.5f)
    .setDuration(250)
    .setInterpolator(new FastOutLinearInInterpolator());

```

Примечание: вызов **start()** для объекта **ViewPropertyAnimator** **НЕ является обязательным**. Если вы этого не сделаете, то просто позволите платформе обработать запуск анимации в соответствующее время (при следующем проходе обработки анимации). Если же вы действительно вызовете **start()**, то будете уверены, что анимация будет запущена немедленно.

81.7. Анимация встряхивания ImageView

В папке **res** создайте новую папку “**anim**” для хранения ресурсов анимации и поместите в нее следующее.

shakeanimation.xml:

```

<?xml version="1.0" encoding="utf-8"?>
<rotate xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="100"
    android:fromDegrees="-15"
    android:pivotX="50%"
    android:pivotY="50%"
    android:repeatCount="infinite"
    android:repeatMode="reverse"
    android:toDegrees="15" />

```

Создайте пустую активность с названием Landing.

activity_landing.xml:

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:id="@+id/imgBell"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:src="@mipmap/ic_notifications_white_48dp"/>

</RelativeLayout>
```

...и метод анимации вида Landing.java:

```
Context mContext;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mContext = this;
    setContentView(R.layout.activity_landing);

    AnimateBell();
}

public void AnimateBell() {
    Animation shake = AnimationUtils.loadAnimation(mContext, R.anim.
    shakeanimation);
    ImageView imgBell = (ImageView) findViewById(R.id.imgBell);
    imgBell.setImageResource(R.mipmap.ic_notifications_active_white_48dp);
    imgBell.setAnimation(shake);
}
```

Глава 82. Расположение

API определения местоположения в Android используются в самых разных приложениях для различных целей, таких как определение местоположения пользователя, уведомление о том, что пользователь покинул определенную область (Geofencing), а также для интерпретации активности пользователя (ходьба, бег, вождение и т.д.).

Однако Android Location API не является единственным средством получения информации о местоположении пользователя. Далее будут приведены примеры использования LocationManager и других распространенных библиотек определения местоположения.

82.1. API Fused Location

Рассмотрим пример использования активности с запросом местоположения:

```
/*
 * Этот пример полезен, если вы хотите получать обновления только в этой
 * активности и не использовать местоположение где-либо еще
 */
```

```
public class LocationActivity extends AppCompatActivity implements
    GoogleApiClient.ConnectionCallbacks, GoogleApiClient.
OnConnectionFailedListener, LocationListener {

    private GoogleApiClient mGoogleApiClient;
    private LocationRequest mLocationRequest;

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

        mGoogleApiClient = new GoogleApiClient.Builder(this)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .addApi(LocationServices.API)
            .build();
    }

    private void requestLocationUpdates() {
        mLocationRequest = new LocationRequest()
            .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY) //GPS качество
точек определения положения
            .setInterval(2000) //По крайней мере, раз в 2 секунды
            .setFastestInterval(1000); //Максимально раз в секунду
    }

    @Override
    protected void onStart(){
        super.onStart();
        mGoogleApiClient.connect();
    }

    @Override
    protected void onResume(){
        super.onResume();
        //Проверка разрешений
        if(ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
            if(mGoogleApiClient.isConnected()) {
                LocationServices.FusedLocationApi.
requestLocationUpdates(mGoogleApiClient, mLocationRequest, this);
            }
        }
    }

    @Override
    protected void onPause(){
        super.onPause();
        //Проверка разрешений
        if(ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
            if(mGoogleApiClient.isConnected()) {
                LocationServices.FusedLocationApi.
removeLocationUpdates(mGoogleApiClient, this);
            }
        }
    }

    @Override
    protected void onStop(){}
```

```

super.onStop();
mGoogleApiClient.disconnect();
}

@Override
public void onConnected(@Nullable Bundle bundle) {
    if(ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
        LocationServices.FusedLocationApi.
requestLocationUpdates(mGoogleApiClient, mLocationRequest, this);
    }
}

@Override
public void onConnectionSuspended(int i) {
    mGoogleApiClient.connect();
}

@Override
public void onConnectionFailed(@NonNull ConnectionResult connectionResult) {
}

@Override
public void onLocationChanged(Location location) {
    //Обработайте здесь код обновления местоположения
}
}

```

Пример использования службы с PendingIntent и BroadcastReceiver

Пример активности (рекомендуемое чтение: <https://developer.android.com/reference/android/support/v4/content/LocalBroadcastManager.html>):

```

/*
 * Этот пример полезен, если у вас есть много различных классов, которые должны
 * получать обновления местоположения, но вы хотите детально контролировать, какие
 * из них будут слушать обновления.
 *
 * Например, эта активность перестанет получать обновления, если она не видна,
 * но база данных класса с зарегистрированным локальным приемником будет продолжать
 * получать обновления, пока не будет вызвана функция "stopUpdates()".
 */
public class ExampleActivity extends AppCompatActivity {

    private InternalLocationReceiver mInternalLocationReceiver;

    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);

        //Создание внутреннего объекта приемника только в этом методе.
        mInternalLocationReceiver = new InternalLocationReceiver(this);
    }

    @Override
    protected void onResume(){
        super.onResume();
        //Регистрация для получения обновлений в активности только тогда,
        когда активность видна
    }
}

```

```

        LocalBroadcastManager.getInstance(this);
registerReceiver(mInternalLocationReceiver, new IntentFilter("googleLocation"));
    }

@Override
protected void onPause(){
    super.onPause();

    //Снятие с регистрации для прекращения получения обновлений в активности,
когда она не видна.
    //Примечание: вы будете получать обновления, даже если эта активность будет
завершена.
    LocalBroadcastManager.getInstance(this).
unregisterReceiver(mInternalLocationReceiver);
}

//Хелпер-метод для получения обновлений
private void requestUpdates(){
    startService(new Intent(this, LocationService.class).putExtra("request",
true));
}

//Хелпер-метод для остановки обновлений
private void stopUpdates(){
    startService(new Intent(this, LocationService.class).putExtra("remove",
true));
}

/*
 * Внутренний приемник, используемый для получения
 * обновлений местоположения для данной активности.
 * Этот приемник и любой приемник, зарегистрированный в LocalBroadcastManager,
 * не нужно регистрировать в манифесте.
 */
private static class InternalLocationReceiver extends BroadcastReceiver{

    private ExampleActivity mActivity;

    InternalLocationReceiver(ExampleActivity activity){
        mActivity = activity;
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        final ExampleActivity activity = mActivity;
        if(activity != null) {
            LocationResult result = intent.getParcelableExtra("result");
            //Обработка обновления местоположения
        }
    }
}
}

```

LocationService

Внимание: не забудьте зарегистрировать эту службу в манифесте!

```

public class LocationService extends Service implements
GoogleApiClient.ConnectionCallbacks, GoogleApiClient.

```

```
OnConnectionFailedListener {  
  
    private GoogleApiClient mGoogleApiClient;  
    private LocationRequest mLocationRequest;  
  
    @Override  
    public void onCreate(){  
        super.onCreate();  
        mGoogleApiClient = new GoogleApiClient.Builder(this)  
            .addConnectionCallbacks(this)  
            .addOnConnectionFailedListener(this)  
            .addApi(LocationServices.API)  
            .build();  
  
        mLocationRequest = new LocationRequest()  
            .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY) //GPS качество  
точек определения положения  
            .setInterval(2000) //По крайней мере, раз в 2 секунды  
            .setFastestInterval(1000); //Максимально раз в секунду  
    }  
  
    @Override  
    public int onStartCommand(Intent intent, int flags, int startId){  
        super.onStartCommand(intent, flags, startId);  
        //Проверка разрешений  
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {  
            if (intent.getBooleanExtra("request", false)) {  
                if (mGoogleApiClient.isConnected()) {  
                    LocationServices.FusedLocationApi.  
requestLocationUpdates(mGoogleApiClient, mLocationRequest, getPendingIntent());  
                } else {  
                    mGoogleApiClient.connect();  
                }  
            }  
            else if(intent.getBooleanExtra("remove", false)){  
                stopSelf();  
            }  
        }  
        return START_STICKY;  
    }  
  
    @Override  
    public void onDestroy(){  
        super.onDestroy();  
        if(mGoogleApiClient.isConnected()){  
            LocationServices.FusedLocationApi.  
removeLocationUpdates(mGoogleApiClient, getPendingIntent());  
            mGoogleApiClient.disconnect();  
        }  
    }  
  
    private PendingIntent getPendingIntent(){  
        //Пример для IntentService  
        //return PendingIntent.getService(this, 0, new Intent(this, **Ваш_INTENT_SERVICE_CLASS_здесь**), PendingIntent.FLAG_UPDATE_CURRENT);  
    }  
}
```

```

//Пример для BroadcastReceiver
return PendingIntent.getBroadcast(this, 0, new Intent(this,
LocationReceiver.class), PendingIntent.FLAG_UPDATE_CURRENT);
}

@Override
public void onConnected(@Nullable Bundle bundle) {
    //Проверка разрешений
    if(ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_
LOCATION) == PackageManager.PERMISSION_GRANTED) {
        LocationServices.FusedLocationApi.
requestLocationUpdates(mGoogleApiClient, mLocationRequest, getPendingIntent());
    }
}

@Override
public void onConnectionSuspended(int i) {
    mGoogleApiClient.connect();
}

@Override
public void onConnectionFailed(@NonNull ConnectionResult connectionResult) {
}

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

```

LocationReceiver

Внимание: не забудьте зарегистрировать этот приемник в манифесте!

```

public class LocationReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        if(LocationResult.hasResult(intent)){
            LocationResult locationResult = LocationResult.extractResult(intent);
            LocalBroadcastManager.getInstance(context).sendBroadcast(new
                Intent("googleLocation").putExtra("result", locationResult));
        }
    }
}

```

82.2. Получение адреса из местоположения с помощью геокодера

Получив из FusedAPI объект Location, вы можете легко получить из него информацию об адресе:

```

private Address getCountryInfo(Location location) {
    Address address = null;
    Geocoder geocoder = new Geocoder(getActivity(), Locale.getDefault());
    String errorMessage;
    List<Address> addresses = null;

```

```

try {
    addresses = geocoder.getFromLocation(
        location.getLatitude(),
        location.getLongitude(),
        // В данном примере получаем только один адрес.
        1);
} catch (IOException ioException) {
    // Перехватываем сетевые или другие проблемы ввода-вывода.
    errorMessage = "IOException>>" + ioException.getMessage();
} catch (IllegalArgumentException illegalArgumentException) {
    // Перехватываем недопустимые значения широты и долготы.
    errorMessage = "IllegalArgumentException>>" + illegalArgumentException.
        getMessage();
}
if (addresses != null && !addresses.isEmpty()) {
    address = addresses.get(0);
}
return country;
}

```

82.3. Запрос обновлений местоположения с помощью LocationManager

Как обычно, необходимо убедиться в наличии необходимых разрешений:

```

public class MainActivity extends AppCompatActivity implements LocationListener{

    private LocationManager mLocationManager = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);

        mLocationManager = (LocationManager) getSystemService(Context.LOCATION_
SERVICE);
    }

    @Override
    protected void onResume() {
        super.onResume();

        try {
            mLocationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
0, 0, this);
        }
        catch(SecurityException e){
            // Приложение не имеет правильных разрешений
        }
    }

    @Override
    protected void onPause() {
        try{
            mLocationManager.removeUpdates(this);
        }

```

```

    catch (SecurityException e){
        // Приложение не имеет правильных разрешений
    }

    super.onPause();
}

@Override
public void onLocationChanged(Location location) {
    // Мы получили обновление местонахождения!
    Log.i("onLocationChanged", location.toString());
}

@Override
public void onStatusChanged(String provider, int status, Bundle extras) {

}

@Override
public void onProviderEnabled(String provider) {

}

@Override
public void onProviderDisabled(String provider) {

}
}

```

82.4. Запрос обновлений местоположения в отдельном потоке с использованием LocationManager

Как обычно, необходимо убедиться в наличии необходимых разрешений:

```

public class MainActivity extends AppCompatActivity implements LocationListener{

    private LocationManager mLocationManager = null;
    HandlerThread mLocationHandlerThread = null;
    Looper mLocationHandlerLooper = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);

        mLocationManager = (LocationManager) getSystemService(Context.LOCATION_
SERVICE);
        mLocationHandlerThread = new HandlerThread("locationHandlerThread");
    }

    @Override
    protected void onResume() {
        super.onResume();

        mLocationHandlerThread.start();
        mLocationHandlerLooper = mLocationHandlerThread.getLooper();
    }
}

```

```
try {
    mLocationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
    0, 0, this, mLocationHandlerLooper);
}
catch(SecurityException e){
    // Приложение не имеет правильных разрешений
}
}

@Override
protected void onPause() {
    try{
        mLocationManager.removeUpdates(this);
    }
    catch (SecurityException e){
        // Приложение не имеет правильных разрешений
    }
}

mLocationHandlerLooper = null;
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN_MR2) {
    mLocationHandlerThread.quitSafely();
} else
    mLocationHandlerThread.quit();

mLocationHandlerThread = null;

super.onPause();
}

@Override
public void onLocationChanged(Location location) {
    // Мы получили обновление местонахождения на отдельном потоке
    Log.i("onLocationChanged", location.toString());
    // // Проверить, в каком потоке вы находитесь, можно следующим образом:
    // Log.d("Какой поток?", Thread.currentThread() == Looper.getMainLooper().
    getThread() ? "UI поток" : "Новый поток");
}

@Override
public void onStatusChanged(String provider, int status, Bundle extras) {
}

@Override
public void onProviderEnabled(String provider) {
}

@Override
public void onProviderDisabled(String provider) {
}
```