

ПРОГРАММИРОВАНИЕ МОБИЛЬНЫХ
ПРИЛОЖЕНИЙ ПОД ПЛАТФОРМУ

ANDROID



Урок № 8

Воспроизведение
аудиофайлов в Android
приложениях.

Класс android.media.
SoundPool

Содержание

1. Воспроизведение аудиофайлов в Android приложениях.	
Класс android.media.SoundPool	4
2. Хранение данных предпочтений.	
Интерфейс android.content.SharedPreferences.....	44
3. Работа с графикой в Android приложениях.....	53
3.1. Классы android.graphics.Canvas, android.graphics.Paint. Событие View.onDraw.	
Метод View.invalidate()	53
3.2. Пример создания своего виджета с помощью android.graphics.Canvas.....	77
3.3. Пример приложения с динамически обновляющейся графикой с использованием метода invalidate() и события onDraw()	108
4. Основы работы с датчиками устройства на примере датчика акселерометра.....	123

4.1. Класс android.hardware.SensorManager	125
4.2. Получение данных с датчика акселерометра	133
5. Домашнее задание	144

Материалы урока прикреплены к данному PDF-файлу. Для доступа к материалам, урок необходимо открыть в программе Adobe Acrobat Reader.

1. Воспроизведение аудиофайлов в Android приложениях. Класс android.media.SoundPool

Класс `android.media.SoundPool` (<https://developer.android.com/reference/android/media/SoundPool>) предназначен для воспроизведения аудиофайлов, которые могут размещаться как в ресурсах приложения, так и за его пределами. Такими аудиофайлами могут быть звуки, являющиеся неотъемлемой частью приложения. Например, звуки коллизий (столкновений) для игровых приложений, сигналы для будильника или уведомлений, звуки реакции на касание в UI приложениях и так далее. Объект `android.media.SoundPool` позволяет воспроизводить одновременно несколько аудиофайлов, причем с возможностью задания каждому воспроизводимому аудиофайлу приоритета. Это отличает объект `android.media.SoundPool` от объекта `android.media.MediaPlayer` (<https://developer.android.com/reference/android/media/MediaPlayer>).

Хотя в официальной документации нет никаких упоминаний об ограничениях ни по размеру ни по длине аудиофайлов для `android.media.SoundPool`, чаще всего объект `android.media.SoundPool` используют для воспроизведения аудиофайлов небольшой продолжительности. И эти аудиофайлы, еще раз повторим, являются неотъемлемой частью работы приложения. Конечно же, правильно такие аудиофайлы размещать именно в ресурсах

приложения, а не за его пределами. Потому что приложению легче управлять такими файлами, они не могут быть случайно потеряны или намеренно заменены, что навредит работе приложения.

Класс `android.media.SoundPool` может загружать аудиофайлы как из каталога assets приложения, так и из каталога /res/raw. В этом разделе мы рассмотрим оба способа загрузки аудиофайлов для воспроизведения.

Давайте детально познакомимся с классом `android.media.SoundPool`. Иерархия класса имеет следующий вид:

```
java.lang.Object
  |
  +-- android.media.SoundPool
```

В классе `android.media.SoundPool` представляют интерес следующие методы:

- `int load(Context context, int resId, int priority)` — метод предназначен для загрузки аудиофайла из каталога ресурсов /res/raw. Метод принимает ссылку на объект контекста приложения (параметр `context`), идентификатор ресурса файла `resId` и приоритет (параметр `priority`) который задает приоритет звука. В настоящее время параметр `priority` игнорируется, поэтому необходимо использовать для этого параметра значение 1 для совместимости с будущими версиями API. Метод возвращает звуковой идентификатор (`soundID`). Значение этого идентификатора `soundID` необходимо сохранить, чтобы использовать для воспроизведения и выгрузки звука.

- `int load(String path, int priority)` — метод предназначен для загрузки аудиофайла из каталога, который не является каталогом ресурсов приложения. Это может быть каталог, находящийся на внутреннем или на внешнем носителе устройства. Метод принимает путь и имя аудиофайла (параметр `path`) и приоритет (параметр `priority`) который задает приоритет звука. В настоящее время параметр `priority` игнорируется, поэтому необходимо использовать для этого параметра значение 1 для совместимости с будущими версиями API. Метод возвращает звуковой идентификатор (`soundID`). Значение этого идентификатора `soundID` необходимо сохранить, чтобы использовать для воспроизведения и выгрузки звука.
- `int load(AssetFileDescriptor afd, int priority)` — метод предназначен для загрузки аудиофайла из каталога ресурсов `assets`. Метод принимает дескриптор файла из каталога `assets` (параметр `afd`) и приоритет (параметр `priority`). В настоящее время параметр `priority` игнорируется, поэтому необходимо использовать для этого параметра значение 1 для совместимости с будущими версиями API. Метод возвращает звуковой идентификатор (`soundID`). Значение этого идентификатора `soundID` необходимо сохранить, чтобы использовать для воспроизведения и выгрузки звука.
- `int play(int soundID, float leftVolume, float rightVolume, int priority, int loop, float rate)` — метод воспроизводит ранее загруженный аудиофайл. Параметр `soundID` идентифицирует аудиофайл, который необходимо

воспроизвести. Значение этого параметра возвращается методом `load()` при загрузке аудиофайла. Хотим обратить ваше внимание на то, что вызов метода `play()` может привести к тому, что другой аудиофайл перестанет воспроизводиться, если превышено максимальное количество одновременно воспроизводимых аудиофайлов. Параметры `leftVolume` и `rightVolume` задают уровень громкости воспроизводимого аудиофайла (в диапазоне от 0.0 до 1.0) для левого и правого стереодинамиков соответственно. Параметр `priority` задает приоритет воспроизведения аудиофайла (значение 0 является значением для минимального приоритета). Параметр `priority` влияет на порядок, в котором потоки будут повторно использоваться для воспроизведения новых звуков. Параметр `loop` задает количество раз повтора воспроизведения аудиофайла. Значение 0 означает, что повторов нет, а значение -1 означает неограниченное количество раз повтора воспроизведения аудиофайла. Параметр `rate` управляет скоростью воспроизведения звука из аудиофайла. Значение 1.0 означает воспроизведение звука с исходной скоростью. Значение 2.0 означает воспроизведение в два раза быстрее, а значение 0,5 означает воспроизведение с половинной скоростью. Значение для параметра `rate` можно задавать в диапазоне от 0.5 до 2.0. Метод `play()` возвращает идентификатор потока воспроизведения звука (`streamID`) или 0 в случае ошибки. Идентификатор потока воспроизведения может использоваться для управления параметрами воспроизведения звука (см. методы ниже).

- `void pause(int streamID)` — метод предназначен для приостановки воспроизведения звука. Метод приостанавливает поток воспроизведения, который указан в параметре `streamID`. Значение `streamID` возвращается методом `play()`. Если поток воспроизводится, он будет приостановлен. Если поток не воспроизводится (например, остановлен или уже был ранее приостановлен), вызов этого метода не будет иметь никакого эффекта. Для возобновления воспроизведения звука необходимо использовать метод `resume()`.
- `void resume(int streamID)` — метод возобновляет воспроизведение ранее приостановленного при помощи метода `pause()` звукового потока `streamID`. Значение `streamID` возвращается методом `play()`. Если поток ранее не был приостановлен, вызов этого метода не будет иметь никакого эффекта.
- `void setLoop(int streamID, int loop)` — метод устанавливает количество раз повтора воспроизведения звукового потока `streamID`. Параметр `loop` задает количество раз повтора воспроизведения аудиофайла. Значение 0 означает, что повторов нет, а значение -1 означает неограниченное количество раз повтора воспроизведения аудиофайла.
- `void setRate(int streamID, float rate)` — метод устанавливает скорость воспроизведения звукового потока `streamID`. Параметр `rate` управляет скоростью воспроизведения звука из аудиофайла. Значение 1.0 означает воспроизведение звука с исходной скоростью. Значение 2.0 означает воспроизведение в два раза быстрее,

а значение 0,5 означает воспроизведение с половинной скоростью. Значение для параметра `rate` можно задавать в диапазоне от 0.5 до 2.0.

- `void setVolume(int streamID, float leftVolume, float rightVolume)` — метод устанавливает громкость воспроизведения звукового потока `streamID`. Значение `streamID` возвращается методом `play()`. Параметры `leftVolume` и `rightVolume` задают уровень громкости воспроизводимого аудиофайла (в диапазоне от 0.0 до 1.0) для левого и правого стереодинамиков соответственно.
- `void stop(int streamID)` — метод останавливает поток воспроизведения `streamID`. Если поток воспроизводится, он будет остановлен. Метод также освобождает все ресурсы, связанные с этим потоком. Если поток не воспроизводится, вызов этого метода не будет иметь никакого эффекта.
- `boolean unload(int soundID)` — метод выгружает ранее загруженный при помощи метода `load()` аудиофайл, указанный в параметре `soundID`. Значение параметра `soundID` возвращается методом `load()`. Метод `unload()` возвращает `true`, если звук успешно выгружен, `false`, если звук уже был выгружен.
- `void release()` — метод освобождает все ресурсы, используемые объектом `android.media.SoundPool`. После вызова этого метода, объект `android.media.SoundPool` больше не может использоваться, и ссылка на него должна быть установлена в значение `null`.

Также добавим что, если возникает необходимость остановить сразу все потоки воспроизведения, то для

этой цели предназначен метод `void autoPause()`. Для возобновления воспроизведения всех приостановленных звуковых потоков предназначен метод `void autoRelease()`.

Что касается методов `load()` класса `android.media.SoundPool`, то в официальной документации не указывается, каким способом (синхронно или асинхронно) происходит загрузка аудиофайлов. Однако, есть интерфейс `android.media.SoundPool.OnLoadCompleteListener` (<https://developer.android.com/reference/android/media/SoundPool.OnLoadCompleteListener>), в котором объявлен всего лишь один метод:

```
void onLoadComplete(SoundPool soundPool,  
                    int sampleId, int status);
```

Этот метод вызывается, когда завершилась загрузка аудиофайла. Следовательно, можно сделать предположение, что методы `load()` класса `android.media.SoundPool` выполняют загрузку асинхронно.

Создание объекта `android.media.SoundPool` зависит от номера версии API. До версии API 21 создание объекта `android.media.SoundPool` осуществлялось при помощи конструктора:

```
SoundPool(int maxStreams, int streamType, int srcQuality);
```

который принимает величину максимального количества одновременно воспроизводимых звуковых потоков (параметр `maxStreams`), тип звукового потока (параметр `streamType`) и неиспользуемый параметр `srcQuality` (в котором необходимо передать значение 0).

Начиная с версии API 21, создать объект android.media.SoundPool можно только при помощи объекта android.media.SoundPool.Builder (<https://developer.android.com/reference/android/media/SoundPool.Builder>):

```
java.lang.Object  
|  
+--- android.media.SoundPool.Builder
```

Использование объекта android.media.SoundPool.Builder обусловлено необходимостью конфигурирования объекта android.media.SoundPool. Методы класса android.media.SoundPool.Builder:

- SoundPool.Builder() — конструктор. Создает объект android.media.SoundPoolBuilder со значениями параметров android.media.SoundPool по умолчанию.
- SoundPool build() — метод создает объект android.media.SoundPool.
- SoundPool.Builder setAudioAttributes(AudioAttributes attributes) — метод задает аудио атрибуты (параметры аудио) в виде объекта android.media.AudioAttributes ([https://developer.android.com/reference/android/media/](https://developer.android.com/reference/android/media/AudioAttributes)
[AudioAttributes](https://developer.android.com/reference/android/media/AudioAttributes)). Объекты класса android.media.AudioAttributes содержат в себе коллекцию атрибутов, содержащих информацию о звуковом потоке. Эта информация используется при воспроизведении звука. Атрибуты позволяют приложению указывать больше информации, чем передается в типе звукового потока, позволяя приложению определять: 1) использование звука, то есть «почему» проигрывается звук, для

чего этот звук используется (задается константами `AudioAttributes.USAGE_XXXX`) и 2) тип контента, то есть «что» проигрывается (задается константами `AudioAttributes.CONTENT_TYPE_XXXX`). В примере, который рассматривается в этом разделе, мы будем использовать константы `AudioAttributes.CONTENT_TYPE_MUSIC` и `AudioAttributes.USAGE_MEDIA`. Более подробную информацию о константах `AudioAttributes.USAGE_XXXX` и `AudioAttributes.CONTENT_TYPE_XXXX` можно получить на страничке официального сайта для разработчиков <https://developer.android.com/reference/android/media/AudioAttributes.html>.

- `SoundPool.Builder setMaxStreams(int maxStreams)` — метод устанавливает максимальное количество одновременно воспроизводимых звуковых потоков.

Перед тем как приступить к примеру, который использует возможности класса `android.media.SoundPool`, рассмотрим фрагменты программного кода, в которых демонстрируется создание объекта `android.media.SoundPool` при помощи `android.media.SoundPool.Builder` (Листинг 1.1), а также загрузка аудиофайлов из каталога ресурсов `assets` (Листинг 1.2) и из каталога ресурсов `/res/raw` (Листинг 1.3).

В Листинге 1.1 показано создание объекта `android.media.SoundPool` в зависимости от версии API: при помощи конструктора (для версии API меньше 21) и при помощи объекта `android.media.SoundPool.Builder` (для версии API больше 21).

Листинг 1.1. Создание объекта android.media.SoundPool в зависимости от версии API

```
public class MainActivity extends AppCompatActivity
{
    // ----- Class constants -----
    /**
     * Maximum number of simultaneous audio streams
     *
     * Максимальное количество одновременно
     * воспроизводимых звуковых потоков
     */
    private final static int SOUNDPOOL_MAX_STREAMS = 5;

    // ----- Class members -----
    /**
     * Reference to android.media.SoundPool object
     *
     * Ссылка на объект android.media.SoundPool
     */
    private SoundPool SP;

    ...

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        ...
        // ----- Create SoundPool object -----
        // ----- Создание объекта SoundPool -----
        if (Build.VERSION.SDK_INT >= 21)
        {
            SoundPool.Builder builder =
                new SoundPool.Builder();
            builder.setMaxStreams(MainActivity.
                SOUNDPOOL_MAX_STREAMS);
            AudioAttributes AA =

```

```
        new AudioAttributes.Builder()
            .setUsage(AudioAttributes.USAGE_MEDIA)
            .setContentType(AudioAttributes.CONTENT_TYPE_MUSIC)
            .build();
        builder.setAudioAttributes(AA);
        this.SP = builder.build();
    }
    else
    {
        this.SP = new SoundPool(
            MainActivity.
            SOUNDPOOL_MAX_STREAMS,
            AudioManager.STREAM_MUSIC, 0);
    }
    ...
}
...
}
```

Загрузка аудиофайлов в объект `android.media.SoundPool` из каталога assets показана в Листинге 1.2.

Листинг 1.2. Загрузка аудиофайлов в объект `android.media.SoundPool` из каталога assets

```
try
{
    this.soundID = this.SP.load(getAssets().
                                openFd(file_name_with_extension), 1);
}
catch (IOException e)
{
    ...
}
```

Загрузка аудиофайлов в объект `android.media.SoundPool` из каталога ресурсов `/res/raw` показана в Листинге 1.3.

Листинг 1.3. Загрузка аудиофайлов в объект `android.media.SoundPool` из каталога ресурсов `/res/raw`

```
this.soundID = this.SP.load(this, R.file_name, 1);
```

Теперь перейдем к примеру использования объекта `android.media.SoundPool` на практике. Для этой цели создан модуль «app» в проекте Android Studio, который прилагается к данному уроку. Внешний вид Активности рассматриваемого примера изображен на Рис. 1.1.

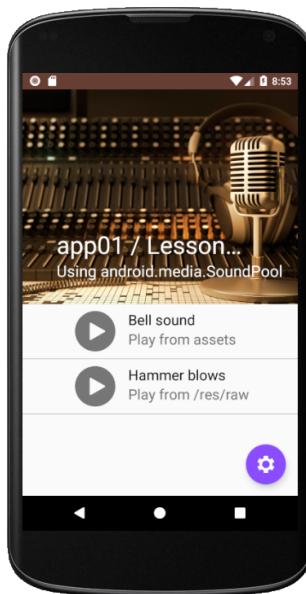


Рис. 1.1. Внешний вид приложения рассматриваемого в данном разделе примера

Как видно из Рис. 1.1, в приложении будут воспроизводиться два звуковых файла: аудиофайл со звуком корабельного колокола (далее будет использоваться название «Bell Sound») и аудиофайл со звуком ударов молотка (далее будет использоваться название «Hammer Blows»). Причем файл со звуком корабельного колокола носит имя bell_sound.mp3 и размещается в каталоге assets приложения, а файл со звуком ударов молотка носит имя hammer.blows.mp3 и размещается в каталоге /res/raw. Эти аудиофайлы взяты из сети Интернет из открытых источников.

Внешний вид Активности из Рис. 1.1 формируется в файле ресурсов /res/layout/activity_main.xml, содержимое которого показано в Листинге 1.4.

Листинг 1.4. Содержимое файла ресурсов /res/layout/activity_main.xml с макетом внешнего вида Активности рассматриваемого в данном разделе примера

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools=
        "http://schemas.android.com/tools"
    xmlns:app=
        "http://schemas.android.com/apk/res-auto"

    android:layout_width="match_parent"
    android:layout_height="match_parent"

    android:fitsSystemWindows="true"
    android:id="@+id/clMain"
    tools:context="com.itstep.myapp.MainActivity">
```

```
<android.support.design.widget.AppBarLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <android.support.design.widget.
        CollapsingToolbarLayout
        android:id="@+id/mainCollapsing"
        android:layout_width="match_parent"
        android:layout_height="300dp"
        app:layout_scrollFlags=
            "scroll|exitUntilCollapsed"
        android:fitsSystemWindows="true"
        app:contentScrim="?attr/colorPrimary"
        app:expandedTitleMarginStart="48dp"
        app:expandedTitleMarginEnd="64dp"
        app:expandedTitleMarginBottom="70dp">

        <ImageView
            android:id="@+id/main.backdrop"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:scaleType="centerCrop"
            android:fitsSystemWindows="true"
            android:src=
                "@drawable/background_music_01"
            app:layout_collapseMode="parallax" />

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textSize="22sp"
            android:textColor=
                "@android:color/white"
            android:layout_marginBottom="32dp"
            android:layout_marginRight="16dp"
            android:layout_marginLeft="48dp"
```

```
    app:layout_collapseMode="parallax"
    android:layout_gravity="bottom"
    android:text=
        "Using android.media.SoundPool"/>

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height=
            "?attr/actionBarSize"
        android:subtitleTextColor="#FFFFFF"
        android:titleTextColor="#FFFFFF"
        app:popupTheme=
            "@style/ThemeOverlay.AppCompat.
            Light"
        app:layout_collapseMode="pin" />

    </android.support.design.widget.
        CollapsingToolbarLayout>

</android.support.design.widget.AppBarLayout>

<android.support.v4.widget.NestedScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior=
        "@string/appbar_scrolling_view_behavior">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
```

```
    android:layout_margin="8dp"
    android:orientation="horizontal">

    <ImageView
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:src=
            "@drawable/playback_button_1"
        android:layout_marginLeft="64dp"
        android:id="@+id/ivOne"
        android:onClick="btnClick" />

    <Space
        android:layout_width="16dp"
        android:layout_height="16dp" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <TextView
            android:layout_width=
                "wrap_content"
            android:layout_height=
                "wrap_content"
            android:layout_gravity="left"
            android:textColor=
                "@color/primary_text"
            android:textSize="20sp"
            android:text="Bell sound" />

        <TextView
            android:layout_width=
                "wrap_content"
            android:layout_height=
                "wrap_content"
```

```
        android:layout_gravity="left"
        android:textColor=
            "@color/secondary_text"
        android:textSize="20sp"
        android:text=
            "Play from assets" />
    </LinearLayout>

</LinearLayout>

<View
    android:layout_width="match_parent"
    android:layout_height="1dp"
    android:background="@color/divider" />

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="8dp"
    android:orientation="horizontal">

    <ImageView
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:src= "@drawable/
            playback_button_1"
        android:layout_marginLeft="64dp"
        android:id="@+id/ivTwo"
        android:onClick="btnClick" />

    <Space
        android:layout_width="16dp"
        android:layout_height="16dp" />

    <LinearLayout
        android:layout_width= "wrap_content"
```

```
        android:layout_height=
                "wrap_content"
        android:orientation="vertical">

        <TextView
            android:layout_width=
                    "wrap_content"
            android:layout_height=
                    "wrap_content"
            android:layout_gravity="left"
            android:textColor=
                    "@color/primary_text"
            android:textSize="20sp"
            android:text="Hammer blows" />

        <TextView
            android:layout_width=
                    "wrap_content"
            android:layout_height=
                    "wrap_content"
            android:layout_gravity="left"
            android:textColor=
                    "@color/secondary_text"
            android:textSize="20sp"
            android:text=
                    "Play from /res/raw" />
        </LinearLayout>
    </LinearLayout>

    <View
        android:layout_width="match_parent"
        android:layout_height="1dp"
        android:background="@color/divider" />

    </LinearLayout>
</android.support.v4.widget.NestedScrollView>
```

```
<android.support.design.widget.  
    FloatingActionButton  
        android:id="@+id/fab"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_gravity="bottom|right"  
        android:layout_marginBottom="16dp"  
        android:layout_marginRight="16dp"  
        android:src="@drawable/fab_configuration"  
        android:onClick="fabClick" />  
  
</android.support.design.widget.CoordinatorLayout>
```

Внешний вид приложения с макетом Активности из Листинга 1.4 изображен на Рис. 1.1.

Как видно из Листинга 1.4, корневым контейнером для Активности является контейнер `android.support.design.widget.CoordinatorLayout` (<https://developer.android.com/reference/android/support/design/widget/CoordinatorLayout>), который предназначен для координации внешнего вида своих дочерних виджетов при изменениях или прокрутках, которые в них происходят. В нашем примере, контейнер `android.support.design.widget.CoordinatorLayout` будет координировать сворачивание или разворачивание панели инструментов (виджет `android.support.v7.widget.Toolbar`) которая размещена в менеджере `android.support.design.widget.CollapsingToolbarLayout` (<https://developer.android.com/reference/android/support/design/widget/CollapsingToolbarLayout>). Эти менеджеры рассматриваются в одном из предыдущих уроков данного курса.

Также на Активности размещены две кнопки (виджеты `android.widget.ImageView`) с идентификаторами `R.id.ivOne` (для воспроизведения звука «Bell Sound») и `R.id.ivTwo` (для воспроизведения звука «Hammer Blows»). Этим виджетам назначен обработчик события нажатия метод `btnClick()`, в котором осуществляется запуск воспроизведения звуковых потоков для «Bell Sound» и «Hammer Blows». Содержимое метода `btnClick()` приведено в Листинге 1.5. Также, в Листинге 1.5 показан код создания объекта `android.media.SoundPool` и код загрузки аудиофайлов «Bell Sound» и «Hammer Blows» из каталогов `assets` и `/res/raw` соответственно.

Листинг 1.5. Класс `MainActivity` рассматриваемого примера с созданием объекта `android.media.SoundPool`, загрузкой в него аудиофайлов «Bell Sound» и «Hammer Blows», а также воспроизведением этих аудиофайлов по событию нажатия на виджеты с идентификаторами `R.id.ivOne` и `R.id.ivTwo`

```
public class MainActivity extends AppCompatActivity
{
    // ----- Class constants -----
    /**
     * Maximum number of simultaneous audio streams
     *
     * Максимальное количество одновременно
     * воспроизводимых звуковых потоков
     */
    private final static int SOUNDPOOL_MAX_STREAMS = 5;

    // ----- Class members -----
    /**
     * Reference to android.media.SoundPool object
     *
```

```
* Ссылка на объект android.media.SoundPool
*/
private SoundPool SP;

/**
 * soundID for audio file bell_sound.mp3
 *
 * Идентификатор soundID для аудиофайла
 * bell_sound.mp3
 */
private int bellSoundID;

/**
 * Audio stream identifier for bell_sound.mp3
 *
 * Идентификатор звукового потока для bell_sound.mp3
 */
private int bellStreamID;

/**
 * Left speaker volume for bell_sound.mp3
 *
 * Громкость левого динамика для bell_sound.mp3
 */
private float bellLeftVolume = 0.5f;

/**
 * Right speaker volume for bell_sound.mp3
 *
 * Громкость правого динамика для bell_sound.mp3
 */
private float bellRightVolume = 0.5f;

/**
 * Playback rate for bell_sound.mp3
 *
```

```
* Скорость воспроизведения для bell_sound.mp3
*/
private float bellRate = 1.0f;

/**
 * soundID for audio file hammer_blow.mp3
 *
 * Идентификатор soundID для аудиофайла
 * hammer_blow.mp3
 */
private int hammerSoundID;

/**
 * Audio stream identifier for hammer_blow.mp3
 *
 * Идентификатор звукового потока для
 * hammer_blow.mp3
 */
private int hammerStreamID;

/**
 * Left speaker volume for hammer_blow.mp3
 *
 * Громкость левого динамика для hammer_blow.mp3
 */
private float hammerLeftVolume = 0.5f;

/**
 * Right speaker volume for hammer_blow.mp3
 *
 * Громкость правого динамика для hammer_blow.mp3
 */
private float hammerRightVolume = 0.5f;

/**
 * Playback rate for hammer_blow.mp3
```

```
*  
* Скорость воспроизведения для hammer_blow.mp3  
*/  
private float hammerRate = 1.0f;  
  
/**  
 * AlertDialog for the configuration of audio streams  
*  
* AlertDialog для конфигурирования параметров  
* звуковых потоков  
*/  
private AlertDialog dialog;  
  
/**  
 * Content view for AlertDialog  
*  
* Виджет с содержимым для AlertDialog  
*/  
private View dialogView;  
  
/**  
 * To round off real numbers (two decimal places)  
*  
* Для округления вещественных чисел (два знака  
* после запятой)  
*/  
private DecimalFormat decimalFormat =  
        new DecimalFormat("##0.00");  
  
// ----- Class methods -----  
@Override  
protected void onCreate(Bundle savedInstanceState)  
{  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
}
```

```
// ----- Toolbar -----
Toolbar toolBar =
        this.findViewById(R.id.toolbar);
this.setSupportActionBar(toolBar);
toolBar.setTitle("app01 / Lesson_08");
toolBar.setTitleTextColor(Color.WHITE);
toolBar.setSubtitleTextColor(Color.WHITE);

// ----- Collapsing Title Text Color -----
CollapsingToolbarLayout mainCollapsing =
        this.findViewById(R.id.mainCollapsing);
mainCollapsing.
        setCollapsedTitleTextColor(Color.WHITE);
mainCollapsing.
        setExpandedTitleColor(Color.WHITE);

// ----- Create SoundPool object -----
// ----- Создание объекта SoundPool -----
if (Build.VERSION.SDK_INT >= 21)
{
    SoundPool.Builder builder =
            new SoundPool.Builder();
    builder.setMaxStreams(MainActivity.
            SOUNDPOOL_MAX_STREAMS);
    AudioAttributes AA =
            new AudioAttributes.Builder()
            .setUsage(AudioAttributes.
            USAGE_MEDIA)
            .setContentType(AudioAttributes.
            CONTENT_TYPE_MUSIC)
            .build();
    builder.setAudioAttributes(AA);

    this.SP = builder.build();
}
else
```

```
{  
    this.SP = new SoundPool(  
        MainActivity.SOUNDPPOOL_MAX_STREAMS,  
        AudioManager.STREAM_MUSIC,  
        0);  
}  
  
/*  
 * Loading audio files to a SoundPool object  
 *  
 * Загрузка аудиофайлов в объект SoundPool  
 * -----  
 */  
// ----- Loading bell_sound.mp3 from assets -----  
// ----- Загрузка аудиофайла bell_sound.mp3  
// ----- из каталога assets -----  
try  
{  
    this.bellSoundID = this.SP.load(  
        getAssets().openFd("bell_sound.mp3"),  
        1);  
}  
catch (IOException e)  
{  
    Toast.makeText(this,  
        "Error loading bell_sound.mp3 : " +  
        e.getMessage(),  
        Toast.LENGTH_LONG).show();  
  
    findViewById(R.id.ivOne).  
        setVisibility(View.INVISIBLE);  
}  
  
// ----- Loading hammer_blow.mp3 from /res/raw  
// ----- Загрузка аудиофайла hammer_blow.mp3  
// ----- из каталога /res/raw -----
```

```
    this.hammerSoundID = this.SP.load(this,
                                         R.raw.hammer_blow, 1);

/*
 * Creating AlertDialog for the configuration
 * of audio streams
 *
 * Создание AlertDialog для конфигурации звуковых
 * ПОТОКОВ -----
 */
//    ***
}

public void btnClick(View v)
{
    switch (v.getId())
    {
// ----- Play bell_sound.mp3 -----
// ----- Воспроизведение bell_sound.mp3 -----
        case R.id.ivOne :
            this.bellStreamID = this.SP.play(
                this.bellSoundID,
                this.bellLeftVolume,
                this.bellRightVolume,
                1,
                0,
                this.bellRate);
            break;

// ----- Play hammer_blow.mp3 -----
// ----- Воспроизведение hammer_blow.mp3 -----
        case R.id.ivTwo :
            this.hammerStreamID = this.SP.play(
                this.hammerSoundID,
                this.hammerLeftVolume,
                this.hammerRightVolume,
                1, 0, this.hammerRate);
    }
}
```

```
        break;
    }
}

public void fabClick(View v)
{
    dialog.show();
}
}
```

Как видно из Листинга 1.5, для каждого из аудиофайлов «Bell Sound» и «Hammer Blows» в классе `MainActivity` объявлены поля (`bellLeftVolume`, `bellRightVolume`, `bellRate`, `hammerLeftVolume`, `hammerRightVolume`, `hammerRate`), которые предназначены для значений уровня громкости левого и правого динамиков, а также для значений скорости воспроизведения (название каждого поля содержит информацию о том, какое значение в нем хранится и для какого аудиофайла это значение предназначено). Это сделано с целью предоставления возможности изменения значения этих характеристик в рассматриваемом приложении.

Для изменения характеристик громкости и скорости воспроизведения звуковых потоков предназначено Диалоговое окно (`android.support.v7.app.AlertDialog`), ссылка на которое объявлена в классе `MainActivity` и называется `dialog`. Внешний вид Диалогового окна с конфигурацией значений громкости и скорости воспроизведения звуковых потоков «Bell Sound» и «Hammer Blows» изображен на Рис. 1.2.

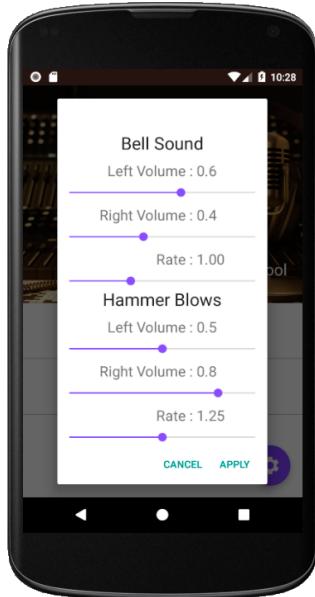


Рис. 1.2. Внешний вид Диалогового окна с конфигурацией
характеристик громкости и скорости воспроизведения
звуковых потоков «Bell Sound» и «Hammer Blows»

Диалоговое окно отображается в приложении при нажатии на кнопку `android.support.design.widget.FloatingActionButton` с идентификатором `R.id.fab` (см. Листинг 1.4). Кнопке `R.id.fab` назначен обработчик события — метод `fabClick()` (см. Листинг 1.5), в котором происходит только отображение Диалогового окна `dialog` при помощи вызова метода `show()`. Создание и настройка Диалогового окна `dialog` осуществляется при помощи объекта `android.support.v7.app.AlertDialog` (<https://developer.android.com/reference/android/support/v7/app/AlertDialog.Builder>) и происходит в методе `onCreate()` класса `MainActivity` (в Листинге 1.5 это место в коде обо-

значено тремя звездочками: ***). Программный код создания Диалогового окна конфигурации звуковых потоков приведен в Листинге 1.6. Макет с набором виджетов внешнего вида Диалогового окна конфигурации находится в файле ресурсов /res/layout/config_dialog.xml и в листингах данного урока не приводится по причине своей понятности. Вы можете посмотреть содержимое этого файла ресурсов в модуле «app» проекта Android Studio, который прилагается к данному уроку. Добавим, что на основе файла с макетом /res/layout/config_dialog.xml создается набор виджетов, на который ссылается ссылка dialogView (см. Листинг 1.6) класса MainActivity.

Листинг 1.6. Программный код создания и настройки Диалогового окна конфигурирования параметров воспроизведения звуковых потоков

```
public class MainActivity extends AppCompatActivity
{
    ...
    /**
     * AlertDialog for the configuration of audio streams
     *
     * AlertDialog для конфигурирования параметров
     * звуковых потоков
     */
    private AlertDialog dialog;

    /**
     * Content view for AlertDialog
     *
     * Виджет с содержимым для AlertDialog
     */
    private View dialogView;
```

```
// ----- Class methods -----
@Override
protected void onCreate(Bundle savedInstanceState)
{
    ...

/*
 * Creating AlertDialog for the configuration
 * of audio streams
 *
 * Создание AlertDialog для конфигурации звуковых
 * потоков
 *
 */
// ----- Create a widget set for the Configuration
// ----- Dialog -----
// ----- Создание набора виджетов для Диалога
// ----- конфигурации -----
    this.dialogView = this.getLayoutInflater().
        inflate(R.layout.config_dialog, null);

// ----- Bell Sound -----
    final TextView tvBellLeftVolume =
        this.dialogView.
            findViewById(R.id.tvBellLeftVolume);
    tvBellLeftVolume.setText(String.
        valueOf(this.bellLeftVolume));

    final TextView tvBellRightVolume =
        this.dialogView.findViewById(R.
            id.tvBellRightVolume);
    tvBellRightVolume.setText(String.
        valueOf(this.bellRightVolume));

    final TextView tvBellRate =
        this.dialogView.findViewById(R.
            id.tvBellRate);
```

```
tvBellRate.setText(String.valueOf(this.  
        bellRate));  
  
// ----- Hammer Blows -----  
final TextView tvHammerLeftVolume =  
    this.dialogView.findViewById(R.  
        id.tvHammerLeftVolume);  
tvHammerLeftVolume.setText(String.  
        valueOf(this.hammerLeftVolume));  
  
final TextView tvHammerRightVolume =  
    this.dialogView.findViewById(R.  
        id.tvHammerRightVolume);  
tvHammerRightVolume.setText(String.  
        valueOf(this.hammerRightVolume));  
  
final TextView tvHammerRate =  
    this.dialogView.findViewById(R.  
        id.tvHammerRate);  
tvHammerRate.setText(String.valueOf(this.  
        hammerRate));  
  
// ----- SeekBar.OnSeekBarChangeListener -----  
SeekBar.OnSeekBarChangeListener SBCL =  
new SeekBar.OnSeekBarChangeListener()  
{  
    @Override  
    public void onProgressChanged(@NonNull  
        SeekBar seekBar,  
        int progress, boolean fromUser)  
    {  
        switch (seekBar.getId())  
        {  
            case R.id.sbBellLeftVolume :  
            {  
                // ----- Bell Sound -----  
                // -----  
            }  
        }  
    }  
}
```

```
        float value = progress / 100f;
        tvBellLeftVolume.setText(
            String.valueOf(value));
    }
    break;

    case R.id.sbBellRightVolume :
    {
        float value = progress / 100f;
        tvBellRightVolume.setText(
            String.valueOf(value));
    }
    break;

    case R.id.sbBellRate :
    {
        float value = 0.5f +
            (progress * 0.015f);
        tvBellRate.setText(
            decimalFormat.
            format(value));
    }
    break;

// ----- Hammer Blows -----
    case R.id.sbHammerLeftVolume :
    {
        float value = progress / 100f;
        tvHammerLeftVolume.setText(
            String.valueOf(value));
    }
    break;

    case R.id.sbHammerRightVolume :
    {
        float value = progress / 100f;
```

```
        tvHammerRightVolume.setText(
            String.valueOf(value));
    }
    break;

    case R.id.sbHammerRate :
    {
        float value = 0.5f +
            (progress * 0.015f);
        tvHammerRate.setText(
            decimalFormat.format(value));
    }
    break;
}
}

@Override
public void onStartTrackingTouch(SeekBar
                                seekBar)
{
}
@Override
public void onStopTrackingTouch(SeekBar
                                seekBar)
{
}
};

// ----- Bell Sound -----
final SeekBar sbBellLeftVolume =
    this.dialogView.findViewById(R.
        id.sbBellLeftVolume);
sbBellLeftVolume.setProgress((int)
    (this.bellLeftVolume * 100));
sbBellLeftVolume.
    setOnSeekBarChangeListener(SBCL);
```

```
final SeekBar sbBellRightVolume =
        this.dialogView.findViewById(R.
                id.sbBellRightVolume);
sbBellRightVolume.setProgress((int)
        (this.bellRightVolume * 100));
sbBellRightVolume.
        setOnSeekBarChangeListener(SBCL);

final SeekBar sbBellRate =
        this.dialogView.findViewById(R.
                id.sbBellRate);
sbBellRate.setProgress((int)((this.bellRate -
        0.5f) / 0.015f));
sbBellRate.setOnSeekBarChangeListener(SBCL);

// ----- Hammer Blows -----
final SeekBar sbHammerLeftVolume =
        this.dialogView.findViewById(R.
                id.sbHammerLeftVolume);
sbHammerLeftVolume.setProgress((int)
        (this.hammerLeftVolume * 100));
sbHammerLeftVolume.
        setOnSeekBarChangeListener(SBCL);

final SeekBar sbHammerRightVolume =
        this.dialogView.findViewById(R.
                id.sbHammerRightVolume);
sbHammerRightVolume.setProgress((int)(this.
        hammerRightVolume * 100));
sbHammerRightVolume.
        setOnSeekBarChangeListener(SBCL);

final SeekBar sbHammerRate =
        this.dialogView.findViewById(R.
                id.sbHammerRate);
sbHammerRate.setProgress((int)((this.
        hammerRate - 0.5f) / 0.015f));
sbHammerRate.setOnSeekBarChangeListener(SBCL);
```

```
// ----- AlertDialog.Builder -----
AlertDialog.Builder builder =
        new AlertDialog.Builder(this,
        android.R.style.
        Theme_DeviceDefault_Light_Dialog);
builder.setView(this.dialogView);
builder.setNegativeButton("Cancel",
        new DialogInterface.OnClickListener()
{
    @Override
    public void onClick(DialogInterface
                        dialog, int which)
    {
// ----- Bell Sound -----
        sbBellLeftVolume. SetProgress(
            (int) (MainActivity.this.
            bellLeftVolume * 100));
        sbBellRightVolume.setProgress(
            (int) (MainActivity.this.
            bellRightVolume * 100));
        sbBellRate.setProgress(
            (int) ((MainActivity.this.
            bellRate - 0.5f) / 0.015f));

// ----- Hammer Blows -----
        sbHammerLeftVolume. SetProgress(
            (int) (MainActivity.this.
            hammerLeftVolume * 100));
        sbHammerRightVolume.setProgress(
            (int) (MainActivity.this.
            hammerRightVolume * 100));
        sbHammerRate.setProgress(
            (int) ((MainActivity.this.
            hammerRate - 0.5f) / 0.015f));
    }
});
```

```
builder.setPositiveButton("Apply",
                           new DialogInterface.OnClickListener()
{
    @Override
    public void onClick(DialogInterface dialog, int which)
    {
// ----- Bell Sound -----
        MainActivity.this.bellLeftVolume =
            sbBellLeftVolume.getProgress() /
            100f;
        MainActivity.this.bellRightVolume =
            sbBellRightVolume.getProgress() /
            100f;
        MainActivity.this.bellRate =
            0.5f + sbBellRate.getProgress() *
            0.015f;

// ----- Hammer Blows -----
        MainActivity.this.hammerLeftVolume =
            sbHammerLeftVolume.getProgress() /
            100f;
        MainActivity.this.hammerRightVolume =
            sbHammerRightVolume.getProgress() /
            100f;
        MainActivity.this.hammerRate =
            0.5f + sbHammerRate.getProgress() *
            0.015f;
    }
});
this.dialog = builder.create();

}
...
}
```

Как видно из Рис 1.2 и Листинга 1.6, для установки значений громкости правого и левого динамика, а также для установки скорости воспроизведения каждого звукового потока используется виджет `android.widget.SeekBar` (<https://developer.android.com/reference/android/widget/SeekBar>) который позволяет менять значение при помощи перетаскивания ползунка или при нажатии пальцем или с помощью клавиш стрелок. Мы ранее не рассматривали этот виджет, но на текущий момент ваших знаний по Android вполне достаточно, чтобы познакомится с этим виджетом заочно. Для виджета `android.widgetSeekBar` можно назначить обработчик события изменения ползунка. Объект обработчика события изменения значения ползунка должен реализовать интерфейс `SeekBar.OnSeekBarChangeListener` (<https://developer.android.com/reference/android/widget/SeekBar.OnSeekBarChangeListener.html>), в котором объявлен метод

```
void onProgressChanged(@NonNull SeekBar seekBar,  
                      int progress, boolean fromUser);
```

который вызывается всякий раз когда происходит изменение позиции ползунка. Этот метод принимает ссылку на `android.widget.SeekBar` (параметр `seekBar`) который является источником события и величину `progress`, которая содержит значение в диапазоне от 0 до 100, соответствующую позиции ползунка в процентах. Параметр `fromUser` будет содержать значение `true` если изменения позиции ползунка осуществил пользователь. В нашем примере

используется обработчик события изменения позиции ползунка для отображения устанавливаемого значения каждой из характеристик звуковых потоков. Например, для громкости правого или левого динамика каждого из звуковых потоков значение, которое может быть установлено при помощи ползунка, может меняться в диапазоне от 0.0 до 1.0. Если перетаскивать ползунок, то при помощи обработчика события `SeekBar.OnSeekBarChangeListener` происходит пересчет позиции ползунка из процентного значения в значение из диапазона от 0.0 до 1.0. Пересчитанное значение тут же отображается в соответствующем виджете `android.widget.TextView`. Например, пересчет значения громкости для левого динамика звукового потока «Hammer Blows» и установка этого значения в соответствующее текстовое поле `android.widget.TextView` (идентификатор `R.id.tvHammerLeftVolume`) показано в Листинге 1.7 (это фрагмент из Листинга 1.6).

Листинг 1.7. Пересчет процентного положения ползунка `android.widgetSeekBar` в значение громкости динамика

```
float value = progress / 100f;  
tvHammerLeftVolume.setText(String.valueOf(value));
```

Аналогично коду, показанному в Листинге 1.7, осуществляется пересчет значения ползунков в величины громкости для правых и левых звуковых динамиков обоих звуковых потоков, которые воспроизводятся в рассматриваемом примере.

Чуть сложнее обстоит ситуация с пересчетом значения ползунка для скорости воспроизведения звукового потока.

Минимальное значение ползунка (т.е. значение 0) должно соответствовать значению скорости 0.5, а максимальная позиция ползунка (т.е. значение 100) должно соответствовать значению скорости 2.0. Пересчет скорости воспроизведения для звукового потока «Bell Sound» и установка этого значения в соответствующее текстовое поле `android.widget.TextView` (идентификатор `R.id.tvBellRate`) показано в Листинге 1.8 (это фрагмент из Листинга 1.6).

Листинг 1.8. Пересчет процентного положения ползунка `android.widgetSeekBar` в значение скорости воспроизведения звукового потока

```
float value = 0.5f + (progress * 0.015f);  
tvBellRate.setText(decimalFormat.format(value));
```

Обратный пересчет из значения скорости воспроизведения (поле класса `MainActivity` — `bellRate`) в процентное значение для положения ползунка показано в Листинге 1.9 (это фрагмент из Листинга 1.6).

Листинг 1.9. Обратный пересчет из значения скорости воспроизведения в процентное значение для положения ползунка `android.widgetSeekBar`

```
final SeekBar sbBellRate =  
    this.dialogView.findViewById(R.id.sbBellRate);  
sbBellRate.setProgress((int)((this.bellRate -  
    0.5f) / 0.015f));
```

Аналогично коду из Листингов 1.8 и 1.9, происходит пересчет для скорости воспроизведения звукового потока «Hammer Blows».

Диалоговое окно конфигурации характеристик звуковых потоков (см. Рис. 1.2) имеет две кнопки: «Cancel» (негативная кнопка, отменяет все изменения, которые осуществил пользователь в Диалоговом окне) и «Apply» (позитивная кнопка, фиксирует все изменения, которые осуществил пользователь в Диалоговом окне). При нажатии на кнопку «Cancel» осуществляется возврат значений ползунков виджетов `SeekBar` в значения, которые соответствуют значениям полей из класса `MainActivity` (то есть полей `bellLeftVolume`, `bellRightVolume`, `bellRate`, `hammerLeftVolume`, `hammerRightVolume`, `hammerRate`). Как это происходит можно посмотреть в обработчике события нажатия на кнопку «Cancel» (метод `onClick()` который назначается при помощи метода `setNegativeButton()`) из Листинга 1.6. При нажатии на кнопку «Apply» происходит запись значений из ползунков в соответствующие поля класса `MainActivity` (то есть поля `bellLeftVolume`, `bellRightVolume`, `bellRate`, `hammerLeftVolume`, `hammerRightVolume`, `hammerRate`). Как это происходит можно посмотреть в обработчике события нажатия на кнопку «Apply» (метод `onClick()` который назначается при помощи метода `setPositiveButton()`) из Листинга 1.6.

Запустите рассматриваемый в данном разделе пример и поэкспериментируйте с разными значениями характеристик воспроизведения звуковых потоков.

Исходный код примера из данного раздела можно найти в модуле «app» проекта Android Studio (version 3.1.2), который прилагается к этому уроку.

2. Хранение данных предпочтений. Интерфейс android.content.SharedPreferences

Из прошлых уроков данного курса вы уже знаете, что приложение может сохранять данные в файлах на внешнем или внутреннем носителе, или в Базе Данных SQLite. Эти способы очень удобны, когда необходимо сохранять большие наборы данных. А что, если приложению необходимо сохранить небольшой набор данных — например параметры конфигурации приложения, или, как их еще называют — «данные предпочтений» (например, громкость воспроизведения звука, цвет фона Активности, название города, для которого приложение запрашивает прогноз погоды и так далее)? Использовать Базу Данных можно, но это весьма тяжелое решение. Использовать файл на внутреннем носителе в данном случае кажется весьма подходящим решением. Однако, есть еще один вариант, который мы рассмотрим в данном разделе — это сохранение данных предпочтений при помощи объекта, который реализует интерфейс `android.content.SharedPreferences` (<https://developer.android.com/reference/android/content/SharedPreferences>). Значения, которые сохраняются в этом объекте, хранятся в виде пары «ключ — значение». В качестве «ключа» выступает строковое значение. В качестве «значения» — значение любого типа. Давайте

рассмотрим методы, которые объявлены в интерфейсе `android.content.SharedPreferences`:

- `boolean contains(String key)` — метод возвращает `true` если в Данных Предпочтений есть значение с ключом `key`.
- `boolean getТип(String key, Тип defValue)` — метод возвращает значение из Данных Предпочтений с ключом `key`. Параметр `defValue` задает значение по умолчанию, которое вернет этот метод в случае, если в Данных Предпочтений нет значения с ключом `key`. Название метода `getТип()` — это собирательное название, которое соответствует методам: `getBoolean()`, `getFloat()`, `getInt()`, `getLong()`, `getString()`.
- `SharedPreferences.Editor edit()` — возвращает ссылку на объект, который реализует интерфейс `SharedPreferences`. `Editor` (<https://developer.android.com/reference/android/content/SharedPreferences.Editor>) при помощи которого можно записывать значения в набор Данных Предпочтений. Другими словами, объект `SharedPreferences`.`Editor` предназначен для редактирования данных которые хранятся в объекте `SharedPreferences`.

Как уже упоминалось выше, для того чтобы записать значение в Данные Предпочтений, используется объект, который реализует интерфейс `SharedPreferences.Editor`. Давайте рассмотрим методы этого интерфейса:

- `void apply()` — метод записывает изменения из этого объекта `SharedPreferences.Editor` в объект `SharedPreferences`, который он редактирует. Вызов этого

метода приводит к записи всех изменений в объект `SharedPreferences`, которые осуществлялись при помощи этого объекта `SharedPreferences.Editor`. В отличие от метода `commit()`, который синхронно записывает измененные данные предпочтений на диск (в постоянное хранилище), метод `apply()` записывает изменения в объект `SharedPreferences` в памяти, и только потом начинает асинхронную запись изменений на диск, при этом, метод `apply()` не возвращает никакой информации о каких-либо сбоях.

- `boolean commit()` — метод аналогичен методу `apply()`. Метод записывает изменения из этого объекта `SharedPreferences.Editor` в объект `SharedPreferences`, который он редактирует. Вызов этого метода приводит к записи всех изменений в объект `SharedPreferences`, которые осуществлялись при помощи этого объекта `SharedPreferences.Editor`. Метод `commit()` сразу же записывает изменения на диск (в постоянное хранилище). Возвращает `true` если изменения сохранились успешно.
- `SharedPreferences.Editor clear()` — метод сообщает редактору `SharedPreferences.Editor` о необходимости удалить все значения из Данных Предпочтений, которые находятся в объекте `SharedPreferences`, который он редактирует. После вызова метода `commit()` единственными оставшимися значениями в объекте `SharedPreferences` будут те значения, которые вы записали в этот редактор. Хотим обратить ваше внимание, что при записи Данных Предпочтений, сначала

выполняется удаление всех значений, независимо от того, как вызывался метод `clear()` — до или после вызова метода `put()`.

- `SharedPreferences.Editor putТип(String key, Тип value)` — метод записывает значение `value` в объект `SharedPreferences` с ключом `key`. Название метода `putТип()` — это собирательное название, которое соответствует методам: `putBoolean()`, `putFloat()`, `.putInt()`, `putLong()`, `putString()`.
- `SharedPreferences.Editor remove(String key)` — метод удаляет значение с ключом `key` из объекта `SharedPreferences`.
- Чтобы получить ссылку на объект, реализующий интерфейс `android.content.SharedPreferences` необходимо воспользоваться методом класса `android.content.Context` (<https://developer.android.com/reference/android/content/Context>):

```
SharedPreferences getSharedPreferences (String name,  
                                     int mode);
```

Этот метод возвращает ссылку на объект, который реализует интерфейс `android.content.SharedPreferences`. Параметр `name` задает файл Данных Предпочтений. Если файл Данных Предпочтений с этим именем не существует, он будет создан при получении редактора (при помощи метода `SharedPreferences.edit ()`). Имя для файла Данных Предпочтений задается разработчиком. Второй параметр `mode` задает режим создания файла. На текущий момент

актуальным является только значение `MODE_PRIVATE` которое равно 0.

Теперь перейдем к рассмотрению примера использование объекта `android.content.SharedPreferences`. Для этого мы добавим в пример из предыдущего раздела (модуль «app») функциональность сохранения и восстановления настроек звуковых потоков в файл Данных Предпочтений.

Вначале добавим в класс `MainActivity` константы с именем файла и с названиями ключей для сохраняемых настроек. В Листинге 2.1 показан программный код объявления этих констант в классе `MainActivity`.

Листинг 2.1. Объявление констант с именем файла Данных Предпочтений и названиями ключей в классе `MainActivity`

```
public class MainActivity extends AppCompatActivity
{
    // ----- Class constants -----
    ...
    /**
     * Filename with Preference Data
     *
     * Имя файла с Данными Предпочтений
     */
    private final static String APP_SHARED_PREFERENCES =
            "my_appl_preferences";

    /*
     * Keys for Shared Preferences
     *
     * Ключи для Данных Предпочтений
     * -----
     */
    private final static String PREF_BELL_LEFT_VOLUME =
            "bellLeftVolume";
```

```

private final static String PREF_BELL_RIGHT_VOLUME =
        "bellRightVolume";
private final static String PREF_BELL_RATE =
        "bellRate";
private final static String PREF_HAMMER_LEFT_VOLUME =
        "hammerLeftVolume";
private final static String PREF_HAMMER_RIGHT_VOLUME =
        "hammerRightVolume";
private final static String PREF_HAMMER_RATE =
        "hammerRate";
...

```

Затем, для сохранения настроек звуковых потоков переопределим в классе `MainActivity` метод `onPause()`, в котором и выполним сохранение настроек в файл Данных Предпочтений. Программный код метода `onPause()` показан в Листинге 2.2.

Листинг 2.2. Сохранение настроек звуковых потоков в файл Данных Предпочтений в методе `onPause()`

```

public class MainActivity extends AppCompatActivity
{
    ...
    @Override
    public void onPause()
    {
        super.onPause();

        // ----- Get the SharedPreferences -----
        // ----- Получение объекта SharedPreferences -----
        SharedPreferences sharedPreferences =
                this.getSharedPreferences(
                        MainActivity.APP_SHARED_PREFERENCES,
                        Context.MODE_PRIVATE);

```

```
SharedPreferences.Editor editor =
    sharedPreferences.edit();
editor.clear();

// ----- Save Bell Sound Preferences -----
// ----- Сохранение настроек для Bell Sound -----
editor.putFloat(
    MainActivity.PREF_BELL_LEFT_VOLUME,
    this.bellLeftVolume);
editor.putFloat(
    MainActivity.PREF_BELL_RIGHT_VOLUME,
    this.bellRightVolume);
editor.putFloat(
    MainActivity.PREF_BELL_RATE,
    this.bellRate);

// ----- Save Hammer Blows Preferences -----
// ----- Сохранение настроек для Hammer Blows ---
editor.putFloat(
    MainActivity.PREF_HAMMER_LEFT_VOLUME,
    this.hammerLeftVolume);
editor.putFloat(
    MainActivity.PREF_HAMMER_RIGHT_VOLUME,
    this.hammerRightVolume);
editor.putFloat(
    MainActivity.PREF_HAMMER_RATE,
    this.hammerRate);
editor.apply();
}

}
```

И наконец, добавим в метод `onCreate()` класса `MainActivity` восстановление настроек звуковых потоков из файла Данных Предпочтений. Программный код восстановления настроек показан в Листинге 2.3.

Листинг 2.3. Восстановление настроек звуковых потоков из файла Данных Предпочтений

```
public class MainActivity extends AppCompatActivity
{
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...

        /*
         * Restore audio stream settings from
         * the Preference Data file
         *
         * Восстановление настроек звуковых потоков
         * из файла Данных Предпочтений
         * -----
         */
        // ----- Get the SharedPreferences -----
        // ----- Получение объекта SharedPreferences -----
        SharedPreferences sharedPreferences =
            this.getSharedPreferences(
                MainActivity.APP_SHARED_PREFERENCES,
                Context.MODE_PRIVATE);

        // ----- Bell Sound -----
        this.bellLeftVolume =
            sharedPreferences.getFloat(
                MainActivity.PREF_BELL_LEFT_VOLUME,
                0.5f);
        this.bellRightVolume =
            sharedPreferences.getFloat(
                MainActivity.PREF_BELL_RIGHT_VOLUME,
                0.5f);
    }
}
```

```
    this.bellRate = sharedPreferences.getFloat(
        MainActivity.PREF_BELL_RATE,
        1.0f);

    // ----- Hammer Blows -----
    this.hammerLeftVolume =
        sharedPreferences.getFloat(
            MainActivity.PREF_HAMMER_LEFT_VOLUME,
            0.5f);
    this.hammerRightVolume =
        sharedPreferences.getFloat(
            MainActivity.PREF_HAMMER_RIGHT_VOLUME,
            0.5f);
    this.hammerRate = sharedPreferences.getFloat(
        MainActivity.PREF_HAMMER_RATE,
        1.0f);

    // ----- Create SoundPool object -----
    // ----- Создание объекта SoundPool -----
    ...
}

...
}
```

Как видно из Листинга 2.3, восстановление настроек звуковых потоков осуществляется перед созданием объекта `android.media.SoundPool`.

Запустите пример из модуля «app» чтобы убедиться, что сохранение и восстановление настроек звуковых потоков при помощи объекта `android.content.SharedPreferences` осуществляется успешно.

Исходный код примера из данного раздела можно найти в модуле «app» проекта Android Studio, который прилагается к этому уроку.

3. Работа с графикой в Android приложениях

В данном разделе мы с вами рассмотрим графические возможности двумерной графики (2D), которые предоставляются разработчикам Android приложений для создания своих (custom) виджетов, для построения графиков и диаграмм и для создания приложений с часто обновляющейся графикой (игровых приложений).

Примечание: Точка начала координат находится в левом верхнем углу. Ось X направлена слева направо, а ось Y направлена сверху вниз (см. Рис. 3.1).



Рис. 3.1. Система координат, которая используется при работе с графикой

3.1. Классы android.graphics.Canvas, android.graphics.Paint. Событие View.onDraw. Метод View.invalidate()

Класс `android.graphics.Canvas` (<https://developer.android.com/reference/android/graphics/Canvas>) представляет из себя поверхность для рисования. Дословно, слово «canvas» переводится с английского как «холст». Любой

виджет (класс `android.view.View` и любой производный от него класс) имеет свою собственную поверхность (холст) для рисования. Эта поверхность используется виджетом для отрисовки своего внешнего вида. Непосредственно, сама отрисовка виджетом своего внешнего вида осуществляется в методе (класс `android.view.View`):

```
void onDraw (Canvas canvas);
```

Операционная система определяет, когда нужно виджету сформировать свой внешний вид и сообщает виджету о необходимости выполнить отрисовку своего внешнего вида путем вызова метода `onDraw()` для этого виджета. При этом, операционная система создает объект `android.graphics.Canvas`, ссылку на который (параметр `canvas`) она передает в метод `onDraw()`. Переданный в метод `onDraw()` объект `android.graphics.Canvas` используется виджетом для рисования своего внешнего вида. Виджет в методе `onDraw()` не создает и не уничтожает объект `android.graphics.Canvas`.

Такой механизм может использоваться разработчиками Android приложений для создания своих собственных (особенных, неповторимых) виджетов. Для этого необходимо создать класс, производный от класса `android.view.View` и переопределить в этом классе метод `onDraw()` как показано в Листинге 3.1. В Листинге 3.1 кроме метода `onDraw()`, который выделен жирным шрифтом, показаны также обязательные конструкты, которые должны присутствовать в классе собственного виджета.

Листинг 3.1. Создание класса для своего (кастомного) виджета

```
class MyView extends View
{
    public MyView(Context context)
    {
        super(context);
    }

    /**
     * Required to override the constructor!
     * Without this constructor, the class will not
     * work !!!
     * Обязательный для переопределения конструктор!
     * Без Этого конструктора класс работать не будет !!!
     */
    public MyView(Context context, AttributeSet attrs)
    {
        super(context, attrs);
    }

    /**
     * Required to override the constructor!
     * Without this constructor, the class will
     * not work !!!
     * Обязательный для переопределения конструктор!
     * Без Этого конструктора класс работать не будет !!!
     */
    public MyView(Context context,
                  AttributeSet attrs, int defStyleAttr)
    {
        super(context, attrs, defStyleAttr);
    }
    @Override
    public void onDraw(Canvas canvas)
    {
```

```

        // Here we draw the appearance of the widget
        // Здесь рисуем внешний вид виджета
    }
}

```

Созданный класс собственного (кастомного) виджета можно смело добавлять в xml файлы ресурсов с версткой макетов внешнего вида, как, например, показано в Листинге 3.2.

Листинг 3.2. Пример применения созданного класса своего виджета в xml верстке макета внешнего вида

```

<package_name.MyView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
/>

```

Сам класс `android.graphics.Canvas` имеет множество методов, которые позволяют рисовать графические примитивы (прямоугольник, окружность, линию и т.д.). Характеристики рисования (такие как цвет и стиль) графических примитивов находятся в объекте `android.graphics.Paint` (<https://developer.android.com/reference/android/graphics/Paint>). Поэтому перед тем, как знакомиться с классом `android.graphics.Canvas`, необходимо познакомимся с классом `android.graphics.Paint`. Иерархия классов для класса `android.graphics.Paint` имеет следующий вид:

```

java.lang.Object
|
+--- android.graphics.Paint

```

Константы класса `android.graphics.Paint` (используются в качестве битовых комбинаций для задания настроек рисования):

- `int ANTI_ALIAS_FLAG` — Флаг, позволяющий осуществлять сглаживание при рисовании. «Антиалиазинг» — это технология, используемая для устранения эффекта «зубчатости», возникающего на краях выводимых на экран плоских или объёмных изображений (см. Рис. 3.2).

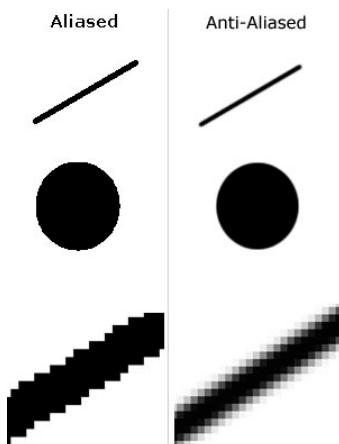


Рис. 3.2. Пример для сглаживания «Anti-Aliasing». Рисунок взят с сайта <https://commons.wikimedia.org/wiki/File:Anti-aliasing.jpg>

- `int DITHER_FLAG` — флаг, осуществляющий «Дизеринг» сглаживание. На сайте <http://wikipedia.org> дано следующее определение «Дизеринга»: «В компьютерной графике дизеринг используется для создания иллюзии глубины цвета для изображений с относительно небольшим количеством цветов в палитре. Отсутству-

ющие цвета составляются из имеющихся путём их «перемешивания». Например, если необходимо получить отсутствующий в палитре фиолетовый цвет, его можно получить, разместив красные и синие пиксели в шахматном порядке; оранжевый цвет может быть составлен из красных и желтых точек».

- `int EMBEDDED_BITMAP_TEXT_FLAG` — флаг, который позволяет использовать растровые шрифты при рисовании текста.
- `int FAKE_BOLD_TEXT_FLAG` — флаг, который применяет синтетический (искусственно созданный) эффект жирного шрифта к выводимому в `android.graphics.Canvas` тексту.
- `int HINTING_OFF` — флаг запрещает «Хинтование» шрифта. На сайте <http://wikipedia.org> дано следующее определение «Хинтования»: «Хинтование это изменение контура шрифта при его растеризации при помощи специальных программных инструкций, заложенных в шрифтовой файл. Используется для обеспечения более чёткого отображения букв на устройствах с низким разрешением экрана или при отображении текста в мелком кегле».
- `int HINTING_ON` — флаг разрешает «Хинтование» шрифта.
- `int LINEAR_TEXT_FLAG` — флаг разрешает плавное линейное масштабирование текста.
- `int STRIKE_THRU_TEXT_FLAG` — флаг, использование которого приводит к выводу текста, все слова которого будут перечеркнуты.

- int **SUBPIXEL_TEXT_FLAG** — флаг включает визуально более четкую отрисовку мелких деталей шрифта.
- int **UNDERLINE_TEXT_FLAG** — флаг, использование которого приводит к выводу текста, все слова которого будут подчеркнуты.

Методы класса `android.graphics.Paint`:

- `Paint()` — конструктор по умолчанию. Создает новый объект `android.graphics.Paint` со значениями по умолчанию.
- `Paint(int flags)` — конструктор, который создет новый объект `android.graphics.Paint` со значениями, которые задаются при помощи битовой комбинации перечисленных выше флагов.
- `Paint(Paint paint)` — конструктор, который создет новый объект `android.graphics.Paint` со значениями из другого объекта `android.graphics.Paint` (параметр `paint`).
- `float measureText(String text)` — возвращает ширину, которую будет занимать при отрисовке текст `text`. При этом учитывается размер шрифта (задается при помощи метода `setTextSize()`) и значения перечисленных выше флагов, которые относятся к характеристикам шрифта.
- `float measureText(String text, int start, int end)` — метод аналогичен предыдущему методу, только рассчитывает ширину части текста `text` между символами с индексами `start` и `end`.
- `void setARGB(int a, int r, int g, int b)` — метод устанавливает значение цвета для рисования в формате ARGB. Значения параметров от 0 до 255.

- `void setAlpha(int a)` — метод устанавливает только величину прозрачности цвета. Значение от 0 до 255.
- `void setAntiAlias(boolean aa)` — метод включает (`true`) или выключает (`false`) режим «Антиалиазинга». Вызов метода аналогичен установке или сбросу флага `Paint.ANTI_ALIAS_FLAG`.
- `void setColor(int color)` — метод устанавливает значение цвета для рисования в виде целочисленного значения. Получить целочисленное значение цвета можно при помощи класса `android.graphics.Color` (<https://developer.android.com/reference/android/graphics/Color>).
- `void setDither(boolean dither)` — метод включает (`true`) или выключает (`false`) режим сглаживания «Дизеринг». Вызов метода аналогичен установке или сбросу флага `Paint.DITHER_FLAG`.
- `void setFlags(int flags)` — метод задает настройки для объекта `android.graphics.Paint` в виде комбинации флагов, которые описаны выше.
- `void setLetterSpacing(float letterSpacing)` — метод задает ширину расстояния между буквами текста. Значение по умолчанию равно 0. Значение задается в единицах «ем».
- `void setTextAlign(Paint.Align align)` — метод задает выравнивание текста (по центру, по правому краю, по левому краю). Значения для выравнивания задаются при помощи перечня `Paint.Align: CENTER, LEFT, RIGHT`. Значение по умолчанию является `LEFT` (выравнивание по левому краю).

- `void setTextScaleX(float scaleX)` — метод задает горизонтальный масштабный коэффициент растяжения или сжатия для текста. Значение по умолчанию равно 1.0. Значения больше 1.0 растянут текст, значения меньше 1.0 будут сужать текст.
- `void setTextSize(float textSize)` — метод устанавливает размер шрифта.
- `Typeface setTypeface(Typeface typeface)` — метод задает (или сбрасывает) для объекта `android.graphics.Paint` объект `android.graphics.Typeface`. Объект класса `android.graphics.Typeface` (<https://developer.android.com/reference/android/graphics/Typeface>) указывает шрифт и стиль шрифта (BOLD, BOLD_ITALIC, ITALIC, NORMAL).
- `void setUnderlineText(boolean underlineText)` — метод устанавливает (`true`) или сбрасывает (`false`) режим подчеркивания текста. Вызов метода аналогичен установке или сбросу флага `Paint.UNDERLINE_TEXT_FLAG`.

Пример использования объекта `android.graphics.Paint` показан в дальнейших Листингах, начиная с Листинга 3.3.

Теперь перейдем к знакомству с классом `android.graphics.Canvas` (<https://developer.android.com/reference/android/graphics/Canvas>). Как уже упоминалось выше, класс является поверхностью для рисования (холстом) и содержит методы для рисования графических примитивов. Фактически, все что рисуется в `android.graphics.Canvas` является рисование растрового изображения (`Bitmap`) и может быть скопировано в объект `android.graphics.Bitmap` (<https://developer.android.com/reference/android/graphics/Bitmap>)

[android/graphics/Bitmap](#)) чтобы, например, сохранить на диск или передать по сети. Возможен и обратный вариант — копирование уже готового объекта `android.graphics.Bitmap` в объект `android.graphics.Canvas` (это часто используется при оптимизации функциональности для приложений с динамически меняющейся графикой).

Итак, иерархия классов для класса `android.graphics.Canvas` выглядит следующим образом:

```
java.lang.Object
  |
  +--- android.graphics.Canvas
```

Методы класса `android.graphics.Canvas`:

- `Canvas(Bitmap bitmap)` — конструктор, создает объект `android.graphics.Canvas` на основе растрового изображения `bitmap`. Все дальнейшее рисование будет осуществляться в переданный объект `android.graphics.Bitmap`. Напомним, что для метода `onDraw()` операционная система сама создает объект `android.graphics.Canvas` и самостоятельно создавать объект `Canvas` в этом случае не нужно. Данный конструктор может понадобится для создания вспомогательного объекта `android.graphics.Canvas` для выполнения специфических действий при сложных графических операциях.
- `void drawArc(float left, float top, float right, float bottom, float startAngle, float sweepAngle, boolean useCenter, Paint paint)` — метод доступен начиная с версии API 21.

Метод предназначен для рисования дуги, которая является частью овала, вписанного в прямоугольник с координатами `left`, `top`, `right`, `bottom`. Параметр `startAngle` задает начальный угол (в градусах), где начинается дуга, а параметр `sweepAngle` задает угол разворота дуги (в градусах), по направлению часовой стрелки. Обратим ваше внимание, что для `startAngle` угол равный 0 градусов соответствует направлению оси X. Параметр `useCenter` если равен `true`, то включает центр дуги в рисуемую фигуру. На Рис 3.2 изображено использование этого метода при рисовании дуги (`startAngle = 90` градусов, `sweepAngle = 120` градусов) с разными значениями параметра `useCenter`. Программный код рисования дуги из Рис. 3.3 показан в Листинге 3.3.

Листинг 3.3. Рисование дуги при помощи метода `drawArc()` класса `android.graphics.Canvas` в методе `onDraw()` класса `MyView` из Листинга 3.1

```
class MyView extends View
{
    ...
    // ----- Class members -----
    private Paint P;
    private RectF RF = new RectF();
    private Bitmap bmp;

    // ----- Anonymous method – for initializing
    // ----- object members -----
    // ----- Безымянный метод для инициализации полей
    // ----- объекта -----
}
```

```
{  
// ----- Create the android.graphics.Paint -----  
// ----- Создаем объект android.graphics.Paint --  
    this.P = new Paint();  
    this.P.setAntiAlias(true);  
}  
  
@Override  
public void onDraw(Canvas canvas)  
{  
// ----- Get the current widget's width and height  
// ----- Получаем текущую высоту и ширину виджета  
    int w = this.getWidth();  
    int h = this.getHeight();  
  
// ----- The background color of our widget -----  
// ----- Цвет фона нашего виджета -----  
    canvas.drawRGB(0xF8, 0xBB, 0xD0);  
  
// ----- drawArc() -----  
    this.P.setColor(Color.rgb(0xC2, 0x18, 0x5B));  
    if (Build.VERSION.SDK_INT >= 21)  
        canvas.drawArc(20, 20, w / 2 - 20,  
                      h / 2 - 20, 90, 120,  
                      true, this.P);  
    else  
    {  
        RF.left = 20;  
        RF.top = 20;  
        RF.right = w / 2 - 20;  
        RF.bottom = h / 2 - 20;  
        canvas.drawArc(this.RF, 90, 120,  
                      true, this.P);  
    }  
}
```

Содержимое файла с макетом внешнего вида Активности /res/layout/activity_main.xml, в котором располагается виджет `MyView` из Листинга 3.3, показано в Листинге 3.4. Виджет `MyView` находится в пакете `itstep.com.myapp2`. В Листинге 3.4 использование этого виджета в файле /res/layout/activity_main.xml выделено жирным шрифтом.

Листинг 3.4. Содержимое файла с макетом внешнего вида Активности /res/layout/activity_main.xml рассматриваемого примера

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:app=
        "http://schemas.android.com/apk/res-auto"
    xmlns:tools=
        "http://schemas.android.com/tools"

    android:layout_width="match_parent"
    android:layout_height="match_parent"

    tools:context=".MainActivity">

    <itstep.com.myapp2.MyView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
    />

</LinearLayout>
```

Как видно из Листинга 3.4, виджет `itstep.com.myapp2.MyView` занимает всю предоставляемую родительским

контейнером область и имеет светло розовый цвет фона, который рисуется в методе `onDraw()` (см. Листинг 3.3).

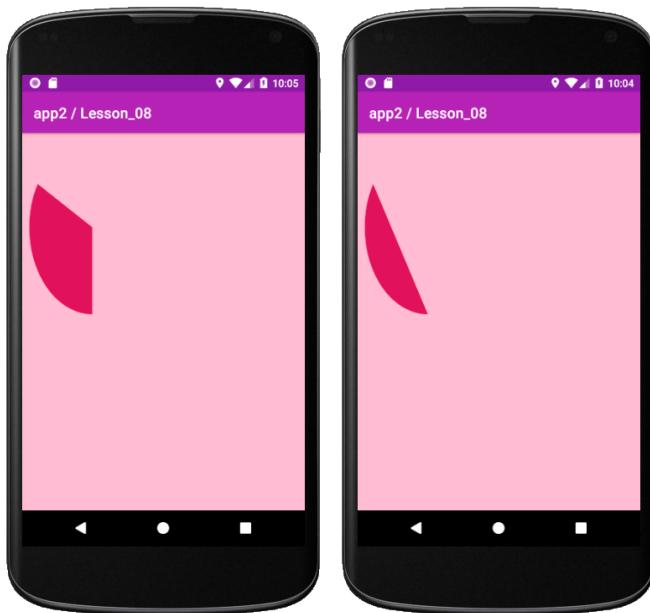


Рис. 3.3. Результат рисование дуги из Листинга 3.3. Слева скриншот со значением параметра `useCenter` равным `true`, справа скриншот со значением параметра `useCenter` равным `false`

Как видно из Листинга 3.3, для версии API до 21 есть другой метод `drawArc()` который принимает координаты и размеры овала для дуги в виде объекта `android.graphics.RectF` (<https://developer.android.com/reference/android/graphics/RectF.html>). Также обратите внимание (см. Листинг 3.3), что в классе `MyView` используется безымянный метод для инициализации объектов, которые участвуют в процессе рисования в методе `onDraw()`. Без-

ымянный метод решает проблему дублирования кода инициализации во всех трех конструкторах (см. Листинг 3.1) класса `MyView`.

Продолжаем знакомиться с методами класса `android.graphics.Canvas`:

- `void drawBitmap(Bitmap bitmap, float left, float top, Paint paint)` — метод рисует в объекте `android.graphics.Canvas` растровое изображение `android.bitmap.Bitmap` (параметр `bitmap`). Изображение `bitmap` рисуется в `android.graphics.Canvas` в координатах, которые задаются параметрами `left` и `top`. Параметр `paint` содержит параметры для копирования (см. флаги класса `android.graphics.Paint`).
- `void drawBitmap(Bitmap bitmap, Rect src, Rect dst, Paint paint)` — метод рисует в объекте `android.graphics.Canvas` растровое изображение `android.bitmap.Bitmap` (параметр `bitmap`). Если быть более точными, то метод копирует часть растрового изображения из параметра `bitmap` в объект `Canvas`. При этом параметр `src` содержит координаты и размеры прямоугольной области, которая будет скопирована, а параметр `dst` содержит координаты и размеры прямоугольной области куда будет скопирована область из `bitmap`. Размеры прямоугольников `src` и `dst` могут быть разными, следовательно, при копировании произойдет масштабирование копируемой области. Параметр `paint` содержит параметры для копирования (см. флаги класса `android.graphics.Paint`). Добавим, что параметр `src` может быть равен `null`. В этом случае будет скопировано все растровое

изображение в прямоугольную область, которая задается параметром `dst`. Давайте рассмотрим пример применения метода `drawBitmap()`. В каталог ресурсов `/res/drawable` поместим изображение (в нашем случае, изображение называется `balls.png`). Добавим в класс `MyView` из Листингов 3.1 и 3.3 поле `bmp` типа `android.graphics.Bitmap`, в безымянном методе инициализации загрузим в это поле изображение `/res/drawable/balls.png` и в методе `onDraw()` выполним отрисовку растрового изображения при помощи метода `drawBitmap()`. Программный код этого примера показан в Листинге 3.5, а результат его работы изображен на Рис. 3.4.



Рис. 3.4. Результат использования метода `drawBitmap()` из Листинга 3.5

Листинг 3.5. Рисование растрового изображения /res/drawable/balls.png при помощи метода drawBitmap() класса android.graphics.Canvas в методе onDraw() класса MyView

```
class MyView extends View
{
    ...
    // ----- Class members -----
    private Paint P;
    private RectF RF = new RectF();
    private Bitmap bmp;
    // ----- Anonymous method – for initializing
    // ----- object members -----
    // ----- Безымянный метод для инициализации полей
    // ----- объекта -----
    {

        ...
        this.bmp = BitmapFactory.decodeResource(
            this.getResources(), R.drawable.balls);
    }

    @Override
    public void onDraw(Canvas canvas)
    {
        ...
        // ----- drawBitmap() -----
        RF.left = w / 2 + 20;
        RF.top = 20;
        RF.right = w - 20;
        RF.bottom = h / 2 - 20;
        canvas.drawBitmap(this.bmp, null, RF, this.P);
    }
}
```

Следующие методы класса android.graphics.Canvas:

- **void drawCircle(float cx, float cy, float radius, Paint paint)** — метод рисует окружность с центром в точке с коор-

динатами `cx` и `cy`, радиусом `radius`, с настройками рисования `paint`.

- `void drawColor (int color)` — метод заполняет весь холст (все растровое изображение) `android.graphics.Canvas` указанным в параметре цветом `color`.
- `void drawLines(float[] pts, Paint paint)` — метод рисует набор линий. Каждая линия задается четырьмя последовательными значениями в массиве `pts`. Первая пара значений задает координаты начала линии, вторая пара значений задает координаты конца линии. Затем следующие четыре значения описывают координаты начала и конца следующей линии и так далее. Настройки рисования задаются в параметре `paint`.
- `void drawOval(float left, float top, float right, float bottom, Paint paint)` — метод доступен начиная с версии API 21. Метод рисует эллипс (овал), используя параметры рисования (цвет и стиль) задаются параметром `paint`. До версии API 21 можно использовать метод `void drawOval (RectF oval, Paint paint)`. Пример рисования эллипса (с учетом версии API) показан в Листинге 3.6, а результат рисования показан на Рис. 3.5.

Листинг 3.6. Рисование эллипса при помощи метода `drawOval()` класса `android.graphics.Canvas` в методе `onDraw()` класса `MyView`

```
// ----- drawOval() -----
this.P.setColor(Color.rgb(0x6A, 0x1B, 0x9A));
if (Build.VERSION.SDK_INT >= 21)
    canvas.drawOval(20, h / 2 + 20,
                    w / 2 - 20, h - 20, this.P);
```

```
else
{
    RF.left = 20;
    RF.top = h / 2 + 20;
    RF.right = w / 2 - 20;
    RF.bottom = h - 20;
    canvas.drawOval(this.RF, this.P);
}
```

Хотим обратить ваше внимание, что использование метода `drawOval()` из версии API 21 (см. Листинг 3.6) более оптимально, т.к. с точки зрения производительности не требует использования дополнительного объекта `android.graphics.RectF` содержащего координаты эллипса.

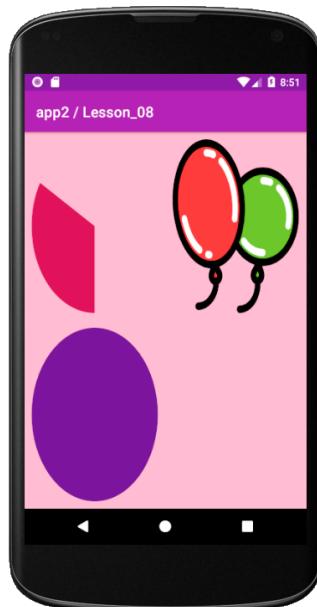


Рис. 3.5. Результат использования метода `drawOval()` из Листинга 3.6

Следующие методы класса `android.graphics.Canvas`:

- `void drawPoint(float x, float y, Paint paint)` — метод рисует точку в координатах `x` и `y`, с цветом, заданным в параметре `paint`.
- `void drawPoints(float[] pts, Paint paint)` — метод рисует множество точек. Каждая точка задается двумя последовательными значениями из массива `pts`. Первое значение — координата по оси `x`, второе — координата по оси `y`. В параметре `paint` задается цвет точек.
- `void drawRGB(int r, int g, int b)` — метод заполняет весь холст (все растровое изображение) `android.graphics.Canvas` цветом, который указан в формате RGB. Значения параметров `r`, `g`, `b` задаются в диапазоне от 0 до 255. Пример применения этого метода можно увидеть в Листинге 3.3.
- `void drawRect(float left, float top, float right, float bottom, Paint paint)` — метод рисует прямоугольник с координатами, которые задаются в параметрах `left`, `top`, `right`, `bottom`. Настройки рисования (цвет и стиль) задаются в параметре `paint`.
- `void drawRoundRect(float left, float top, float right, float bottom, float rx, float ry, Paint paint)` — метод доступен начиная с версии API 21. Он рисует прямоугольник с закругленными углами. Координаты задаются, в параметрах `left`, `top`, `right`, `bottom`. Радиусы закругления углов задаются в параметрах `rx` (радиус закругления по оси x) и `ry` (радиус закругления по оси y). Настройки рисования (цвет и стиль) задаются в параметре `paint`. До версии API 21 можно использовать метод `void`

`drawRoundRect (RectF rect, float rx, float ry, Paint paint)`. Пример рисования прямоугольника с закругленными углами (с учетом версии API) показан в Листинге 3.7, а результат на Рис. 3.6 слева.

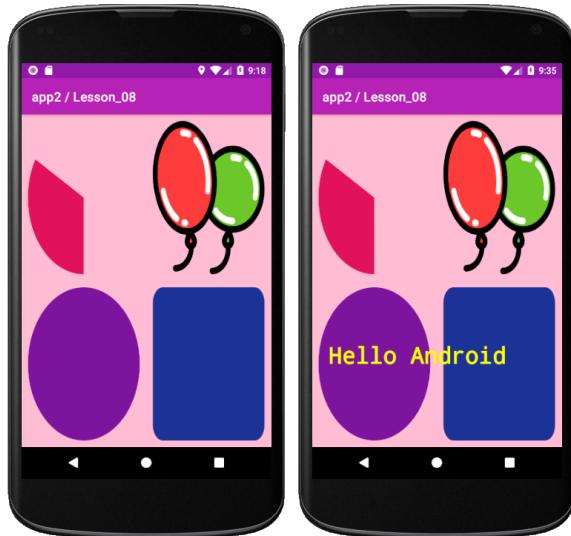


Рис. 3.6. Результат использования метода `drawRoundRect()` из Листинга 3.7 (слева) и результат использования метода `drawText()` из Листинга 3.8 (справа)

- `void drawText(String text, float x, float y, Paint paint)` — метод выводит строку `text`. Координаты начала текста задаются в `x` и `y`. Параметры рисования текста задаются в `paint`. Пример рисования текста (моноширинный шрифт, желтого цвета, размер 70 пикселей) показан в Листинге 3.8. Результат показан на Рис. 3.6 справа.
- `int getHeight()` — метод возвращает высоту растрового изображения (поверхности для рисования) `android.graphics.Canvas`.

- `int getWidth()` — метод возвращает ширину растрового изображения (поверхности для рисования) `android.graphics.Canvas`.

Листинг 3.8. Рисование текста (моноширный шрифт, желтого цвета, размер 70 пикселей) при помощи метода `drawText()` класса `android.graphics.Canvas` в методе `onDraw()` класса `MyView`

```
// ----- drawRoundRect() -----
    this.P.setColor(Color.rgb(0x28, 0x35, 0x93));
    if (Build.VERSION.SDK_INT >= 21)
        canvas.drawRoundRect(w / 2 + 20, h / 2 + 20,
                             w - 20, h - 20, 30, 50,
                             this.P);
    else
    {
        RF.left = w / 2 + 20;
        RF.top = h / 2 + 20;
        RF.right = w - 20;
        RF.bottom = h - 20;
        canvas.drawRoundRect(this.RF, 30, 50, this.P);
    }
```

Листинг 3.7. Рисование прямоугольника с закругленными углами при помощи метода `drawRoundRect()` класса `android.graphics.Canvas` в методе `onDraw()` класса `MyView`

```
// ----- drawText() -----
this.P.setColor(Color.rgb(0xFF, 0xFF, 0x00));
Typeface tf = Typeface.create(Typeface.MONOSPACE,
                             Typeface.BOLD);
this.P.setTextSize(70);
this.P.setTypeface(tf);
canvas.drawText("Hello Android", 50, 3 * h / 4, this.P);
```

Все перечисленные примеры рисовали простейшие геометрические фигуры в виде заполненных указанным в объекте `android.graphics.Paint` цветом. Для случая, когда необходимо нарисовать фигуру контуром, необходимо воспользоваться стилем `android.graphics.Paint.Style` (<https://developer.android.com/reference/android/graphics/Paint.Style>). Это перечень, в котором указаны следующие стили рисования: `FILL` (геометрическая фигура рисуется заполненной), `FILL_AND_STROKE` (геометрическая фигура рисуется заполненной и обводится контуром), `STROKE` (геометрическая фигура рисуется только при помощи контура). Требуемый стиль `Paint.Style` назначается объекту `android.graphics.Paint` при помощи метода `void setStyle(Paint.Style style)`. Пример рисования фигуры (окружности) контуром приведен в Листинге 3.9 и изображен на Рис. 3.7.

Листинг 3.9. Рисование окружности контуром при помощи метода `drawCircle()` класса `android.graphics.Canvas` в методе `onDraw()` класса `MyView`

```
// ----- drawCircle() and Paint.Style = STROKE

this.P.setColor(Color.rgb(0xFF, 0xFF, 0x00));
this.P.setStyle(Paint.Style.STROKE);
this.P.setStrokeWidth(3);
canvas.drawCircle(w / 2, h / 2,
    ((w < h)?w:h) / 2 - 30, this.P);
```

Обратите внимание, что толщина контура (см. Листинг 3.9) задается при помощи метода `void setStrokeWidth(float width)` класса `android.graphics.Paint`.



Рис. 3.7. Пример рисования фигуры (окружности) контуром из Листинга 3.9

И в завершении данного раздела хотим рассказать вам о методе класса `android.view.View`:

```
void invalidate();
```

Метод сообщает операционной системе, что внешний вид виджета (для которого вызывается этот метод) недействителен и требует перерисовки. Если виджет виден, то, как только в системе появится возможность, для этого виджета будет вызван метод `void onDraw(android.graphics.Canvas canvas)`. Метод `invalidate()` должен вызываться из потока пользовательского интерфейса. Для вызова из другого потока, необходимо вызывать метод:

```
void postInvalidate();
```

Примеры использования методов `validate()` и `invalidate()` будут показаны в следующих разделах.

Исходный код примера из данного раздела можно найти в модуле «app2» проекта Android Studio, который прилагается к этому уроку.

3.2. Пример создания своего виджета с помощью android.graphics.Canvas

В этом разделе мы рассмотрим каким образом можно применить графические инструменты класса `android.graphics.Canvas` для создания оригинальных (кастомных) виджетов. Зачем создавать оригинальные виджеты? Дело в том, что не редко возникают ситуации, когда стандартный набор виджетов не содержит необходимого виджета и выходом из этих ситуаций является самостоятельное создание необходимого виджета. Также создание оригинальных виджетов делается с целью сделать разрабатываемое приложение выгодно отличающимся от других приложений с точки зрения внешнего вида и стиля. Ведь каждый разработчик стремится к тому, чтобы его приложение было не похоже на все остальные. Примерами таких виджетов могут быть: виджет настройки громкости, виджет эквалайзера, виджет выбора цвета, виджет выбора даты и времени, и так далее и так далее. Все зависит от фантазии разработчика.

В примере этого раздела будет создаваться виджет выбора цвета. С помощью этого виджета пользователь сможет выбирать цвет из некоторой палитры цветов и, затем, выбранный цвет можно будет применить к каким-то

другим виджетам в качестве цвета фона или текста. Класс создаваемого виджета выбора цвета будет называться `MyColorPicker`. Для примера из этого раздела создан модуль «app3» в проекте Android Studio, который прилагается к данному уроку. Внешний вид виджета `MyColorPicker` изображен на Рис. 3.8. В рассматриваемом примере при помощи виджета выбора цвета `MyColorPicker` будет осуществляться выбор фонового цвета для тулбара `android.support.v7.widget.Toolbar`.

Содержимое файла ресурсов xml с макетом внешнего вида Активности `/res/layout/activity_main.xml` приведено в Листинге 3.10.

Листинг 3.10. Содержимое файла ресурсов xml с макетом внешнего вида Активности `/res/layout/activity_main.xml` рассматриваемого примера

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

    android:orientation="vertical"
    android:id="@+id/llMain"
    tools:context=".MainActivity">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="120dp"
```

```
app:popupTheme=
    "@style/ThemeOverlay.AppCompat.Light"
    android:background="@color/colorPrimary" />

<itstep.com.myapp3.MyColorPicker
    android:layout_width="280dp"
    android:layout_height="280dp"
    android:layout_margin="16dp"
    android:layout_gravity="center_horizontal"
    android:id="@+id/mcpOne" />
</LinearLayout>
```

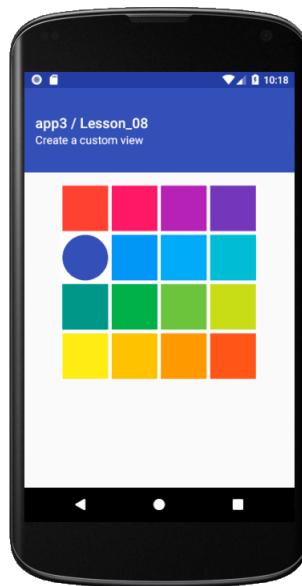


Рис. 3.8. Внешний вид кастомного виджета выбора цвета MyColorPicker, создание которого рассматривается в данном разделе

В Листинге 3.10 на Активности размещается виджет `itstep.com.myapp3.MyColorPicker`. Размеры, которые от-

водятся этому виджету, составляют 280dp по ширине и 280dp по высоте (т.е. виджет имеет форму квадрата). Код создания виджета `itstep.com.myapp3.MyColorPicker` в Листинге 3.10 выделен жирным шрифтом. Внешний вид виджета изображен на Рис. 3.8.

Содержимое класса `MyColorPicker` показано в Листинге 3.11.

Листинг 3.11. Содержимое класса `MyColorPicker`, который является виджетом для выбора цветов

```
class MyColorPicker extends View
{
    /**
     * Current selected color
     *
     * Текущий выбранный цвет
     */
    private int curColor = colors[4];

    /**
     * Set of colors available for selection
     *
     * Набор доступных для выбора цветов
     */
    private static int[] colors =
    {
        Color.rgb(0xF4, 0x43, 0x36),
        Color.rgb(0xE9, 0x1E, 0x63),
        Color.rgb(0x9C, 0x27, 0xB0),
        Color.rgb(0x67, 0x3A, 0xB7),
        Color.rgb(0x3F, 0x51, 0xB5),
        Color.rgb(0x21, 0x96, 0xF3),
        Color.rgb(0x03, 0xA9, 0xF4),
        Color.rgb(0x00, 0xBC, 0xD4),
    }
}
```

```
        Color.rgb(0x00, 0x96, 0x88),  
        Color.rgb(0x4C, 0xAF, 0x50),  
        Color.rgb(0x8B, 0xC3, 0x4A),  
        Color.rgb(0xCD, 0xDC, 0x39),  
        Color.rgb(0xFF, 0xEB, 0x3B),  
        Color.rgb(0xFF, 0xC1, 0x07),  
        Color.rgb(0xFF, 0x98, 0x00),  
        Color.rgb(0xFF, 0x57, 0x22)  
    };  
  
    public MyColorPicker(Context context)  
    {  
        super(context);  
    }  
  
    /**  
     * Required to override the constructor!  
     *  
     * Обязательный для переопределения конструктор!  
     */  
    public MyColorPicker(Context context,  
                        AttributeSet attrs)  
    {  
        super(context, attrs);  
    }  
  
    /**  
     * Required to override the constructor!  
     *  
     * Обязательный для переопределения конструктор!  
     */  
    public MyColorPicker(Context context,  
                        AttributeSet attrs,  
                        int defStyleAttr)  
    {  
        super(context, attrs, defStyleAttr);  
    }
```

```
/**  
 * The method returns the color selected  
 * in the widget  
 *  
 * Метод возвращает выбранный в виджете цвет  
 */  
public int getColor()  
{  
    return this.curColor;  
}  
  
// ----- Class members -----  
private Paint P;  
// ----- Anonymous method – for initializing  
// ----- object members -----  
// ----- Безымянный метод для инициализации полей  
// ----- объекта -----  
{  
// ----- Create the android.graphics.Paint -----  
// ----- Создаем объект android.graphics.Paint  
    this.P = new Paint();  
    this.P.setAntiAlias(true);  
}  
  
@Override  
public void onDraw(Canvas canvas)  
{  
// ----- width of one cell -----  
// ----- ширина одной ячейки -----  
    int w = this.getWidth() / 4;  
  
// ----- height of one cell -----  
// ----- высота одной ячейки -----  
    int h = this.getHeight() / 4;  
  
// ----- Fill the entire surface with  
// ----- a background white color -----
```

```

// ----- Заполняем всю поверхность фоновым белым
// ----- цветом -----
    canvas.drawColor(Color.WHITE);

// ----- Drawing available colors for selection
// ----- in the form of a 4 by 4 grid
// ----- Рисование доступных для выбора цветов
// ----- в виде сетки 4 на 4 -----
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {

// ----- Color from the array of colors for
// ----- the current cell -----
// ----- Цвет из массива цветов для текущей
// ----- клетки -----
            this.P.setColor(colors[i * 4 + j]);
            if (this.curColor == colors[i * 4 + j])
            {

// ----- The current selected color should be
// ----- displayed as a selected color.
// ----- Текущий выбранный цвет нужно отобразить
// ----- как выделенный -----
                if (Build.VERSION.SDK_INT >= 21)
                    canvas.drawOval(5 + j * w,
                        5 + i * h, (j + 1) * w - 5,
                        (i + 1) * h - 5, this.P);
                else
                {
                    canvas.drawRect(15 + j * w,
                        15 + i * h, (j + 1) * w - 15,
                        (i + 1) * h - 15, this.P);
                }
            }
        }
    }
}

```

```
        else
        {
            canvas.drawRect(5 + j * w, 5 + i * h,
                (j + 1) * w - 5,
                (i + 1) * h - 5, P);
        }
    }
}
```

Для того, чтобы отобразить набор доступных для выбора цветов, в классе `MyColorPicker` создан статический массив `colors` (см. Листинг 3.11). В этом массиве объявлено 16 некоторых цветов. Количество цветов подобрано таким образом, чтобы виджет `MyColorPicker` выглядел как сетка размером 4 на 4 ячейки. Всего 16 ячеек. В каждой ячейке будет отображаться один цвет из массива `colors`. Также, в классе `MyColorPicker` объявлено закрытое поле `curColor` (см. Листинг 3.11) которое предназначено для хранения значения цвета, который выбран в виджете `MyColorPicker`. Для получения значения выбранного цвета (значения поля `curColor`) предназначен метод `getColor()`.

В методе `onDraw()` выполняется отрисовка внешнего вида виджета `MyColorPicker`. Первоначально происходит определение ширины ячейки (локальная переменная `w`) как $\frac{1}{4}$ от ширины всего виджета `MyColorPicker`, и высоты ячейки (локальная переменная `h`) как $\frac{1}{4}$ от ширины всего виджета `MyColorPicker`. Затем, при помощи цикла и вложенного в него цикла, происходит рисование сетки с заполненными доступными цветами ячейками.

При этом, перед рисованием каждой ячейки, происходит проверка — является ли текущая рисуемая ячейка ячейкой, которая содержит выбранный в данный момент цвет (т.е. цвет равный значению поля `curColor`)? Если да, то текущую ячейку необходимо нарисовать с некоторым признаком, делающим акцент на то, что эта ячейка содержит выделенный в данный момент цвет, в противном случае — текущую ячейку необходимо рисовать как обычно. В Листинге 3.12 приведен фрагмент этого кода.

Листинг 3.12. Фрагмент кода из Листинга 3.11, показывающий часть метода `onDraw()` с рисованием ячейки содержащей выбранный на данный момент цвет

```
if (this.curColor == colors[i * 4 + j])
{
    // ----- The current selected color should be
    // ----- displayed as a selected color.
    // ----- Текущий выбранный цвет нужно отобразить
    // ----- как выделенный -----
    if (Build.VERSION.SDK_INT >= 21)
        canvas.drawOval(5 + j * w, 5 + i * h,
                        (j + 1) * w - 5, (i + 1) * h - 5, this.P);
    else
    {
        canvas.drawRect(15 + j * w, 15 + i * h,
                        (j + 1) * w - 15, (i + 1) * h - 15, this.P);
    }
}
else
{
    canvas.drawRect(5 + j * w, 5 + i * h,
                    (j + 1) * w - 5, (i + 1) * h - 5, this.P);
}
```

Как видно из Листинга 3.12, ячейка которая содержит выбранный в данный момент цвет, отображается в виде окружности (см. Рис. 3.8) при помощи метода `drawOval()`. Однако, метод `drawOval()` доступен только начиная с версии API 21. Поэтому в программе делается проверка на версию API и если она больше чем API 21, то ячейка содержащая выбранный цвет рисуется в виде окружности, в противном случае, ячейка рисуется в виде прямоугольника, который имеет меньший размер (на 20dp по вертикали и горизонтали), чем другие ячейки. Наверняка вы уже обратили внимание (см. Рис. 3.8), что между всеми ячейками существует отступ. Этот отступ задается следующим образом:

```
canvas.drawRect(5 + j * w, 5 + i * h,  
                (j + 1) * w - 5, (i + 1) * h - 5,  
                this.P);
```

То есть, при рисовании ячейки, прямоугольник (который находится в i -ой строке и j -ом столбце) рисуется с отступом в 5dp сверху и слева, и шириной и высотой меньшей на 5dp. Такой способ предоставляет возможность не «лепить в притык» ячейки, а красиво их расположить с отступом в 10dp. Так вот, для ячейки которая содержит выбранный цвет в случае, если версия API будет меньше чем API 21, отступ будет равен 20dp:

```
canvas.drawRect(15 + j * w, 15 + i * h,  
                (j + 1) * w - 15, (i + 1) * h - 15,  
                this.P);
```

Внешний вид виджета MyColorPicker для версии API меньшей чем API 21 показан на Рис. 3.9.



Рис. 3.9. Внешний вид виджета MyColorPicker для версии API меньшей чем API 21

Итак, сейчас у нас готов программный код класса `MyColorPicker`, который может отрисовать свой внешний вид при помощи переопределения метода `onDraw()`. Если мы запустим приложение, то увидим что виджет выглядит как на Рис. 3.8, но пока не реагирует на нажатия пользователя. Поэтому необходимо сделать следующий шаг — добавить обработчик события нажатия на виджет `MyColorPicker`.

Конечно же, обрабатывать события нажатия пользователя должен непосредственно сам виджет `My-`

ColorPicker. Ведь в качестве результата, у нас должен получиться виджет, готовый к использованию сторонними программистами. Добавим к объявлению класса MyColorPicker реализацию интерфейса View.OnTouchListener (<https://developer.android.com/reference/android/view/View.OnTouchListener>). И в конструкторе класса MyColorPicker зарегистрируем объект MyColorPicker на получение событий нажатий пользователя. То есть, виджет MyColorPicker будет получать события нажатия и сам же и будет их обрабатывать. Однако, в классе MyColorPicker несколько конструкторов. Неудобно, с программной точки зрения, прописывать один и тот же программный код в разные конструкторы. Поэтому мы не будем копировать код регистрации на получение событий нажатия в каждый конструктор, а воспользуемся безымянным методом.

Напомним, что безымянные методы в Java вызываются при создании объекта и эти методы удобно использовать для инициализации объекта в случае, если у класса этого объекта много конструкторов и существует некий единый код инициализации (как в нашем случае). Программный код реализации интерфейса View.OnTouchListener в классе MyColorPicker показан в Листинге 3.13. Также отметим, что выбор в пользу события View.OnTouchListener объясняется тем, что в этом событии можно получить координаты точки нажатия, ведь нашему виджету MyColorPicker необходимо определять какую именно ячейку нажал пользователь для выбора цвета.

Листинг 3.13. Обработка события нажатия View.OnTouchListener в классе MyColorPicker с целью определения выбора цвета

```
class MyColorPicker extends View implements
    View.OnTouchListener
{
    ...
    // ----- Anonymous method -----
    // ----- Безымянный метод -----
    {

        // ----- Register the current object to receive
        // ----- touch events -----
        // ----- Регистрируем текущий объект на получение
        // ----- событий нажатия -----
        this.setOnTouchListener(this);
    }
    ...

    @Override
    public boolean onTouch (View v, MotionEvent event)
    {
        // ----- Width of one cell -----
        // ----- Ширина одной ячейки -----
        int w = this.getWidth() / 4;

        // ----- Height of one cell -----
        // ----- Высота одной ячейки -----
        int h = this.getHeight() / 4;

        // ----- The row to which you clicked -----
        // ----- Стока в которую кликнули -----
        int row = (int) event.getY() / h;

        // ----- The column that was clicked -----
        // ----- Столбец в который кликнули -----
    }
}
```

```
    int col = (int) event.getX() / w;

    // ----- Get the color that the user chose -----
    // ----- Получаем цвет, который выбрал пользователь
    this.curColor = colors[row * 4 + col];

    // ----- System! As soon as possible,
    // ----- call the onDraw () method! -----
    // ----- Система! Как можно быстрее вызови метод
    // ----- onDraw()!
    this.invalidate();
    return true;
}

}
```

Как видно из Листинга 3.13, в обработчике события нажатия `onTouch()` происходит определение по координатам, в какую ячейку нажал пользователь. Ячейка идентифицируется по индексу строки и столбца, в которой она находится. Затем, индексы строки и столбца пересчитываются в индекс цвета из массива цветов. И этот цвет назначается полю объекта `curColor`. После этого виджет `MyColorPicker` необходимо перерисовать. Для этого вызывается метод `invalidate()` класса `android.view.View`. В Листинге 3.13 вызов метода `invalidate()` выделен жирным шрифтом.

Теперь, если мы запустим наше приложение, то увидим, что виджет `MyColorPicker` реагирует на нажатия пользователя и показывает выбранный пользователем цвет (ячейка которая содержит окружность). Остался последний шаг — чтобы при выборе цвета в виджете

MyColorPicker менялся цвет фона тулбара R.id.toolbar (см. Листинг 3.10). Для этого будем использовать механизм оповещения о событиях, который принят в языке программирования Java: источником события будет объект MyColorPicker, получатели событий должны реализовать соответствующий интерфейс (назовем этот интерфейс MyColorPicker.OnColorPickListener) и получатели событий должны зарегистрироваться на получение событий у объекта MyColorPicker. Добавим объявление интерфейса MyColorPicker.OnColorPickListener в класс MyColorPicker. Также добавим в класс MyColorPicker всю необходимую функциональность по регистрации, отмены регистрации и оповещению слушателей о событиях смены цвета. В Листинге 3.14 показаны добавленные в класс MyColorPicker изменения.

Листинг 3.14. Функциональность класса MyColorPicker относящаяся к оповещению событий смены выбранного цвета всем зарегистрированным слушателям

```
class MyColorPicker extends View implements
    View.OnTouchListener
{
    public interface OnColorPickListener
    {
        void onColorPick(int clr);
    }
    /**
     * Collection of listeners of the color change
     * event
     *
     * Коллекция получателей события смены цвета
     */
}
```

```
private ArrayList<OnColorPickListener> listeners =  
    new ArrayList<>();  
  
public void addOnColorPickListener(  
    MyColorPicker.OnColorPickListener listener)  
{  
    this.listeners.add(listener);  
}  
  
public void removeOnColorPickListener(  
    MyColorPicker.OnColorPickListener listener)  
{  
    this.listeners.remove(listener);  
}  
  
@Override  
public boolean onTouch (View v, MotionEvent event)  
{  
    ...  
    // ----- System! As soon as possible,  
    // ----- call the onDraw () method! -----  
    // ----- Система! Как можно быстрее вызови метод  
    // ----- onDraw()! -----  
    this.invalidate();  
  
    // ----- Alerting all listeners about the color  
    // ----- change -----  
    // ----- Оповещение всех слушателей о смене цвета  
    // -----  
    for (int i = 0; i < this.listeners.size(); i++)  
    {  
        this.listeners.get(i).  
            onColorPick(this.curColor);  
    }  
    return true;  
}  
}
```

Как мы можем видеть из Листинга 3.14, в интерфейсе `MyColorPicker.OnColorPickListener` объявлен один метод:

```
void onColorPick(int clr);
```

Который будет вызываться когда пользователь выберет в виджете `MyColorPicker` цвет. Метод `onColorPick()` в параметре `clr` содержит значение выбранного пользователем цвета.

Также в классе `MyColorPicker` объявлена коллекция `listeners` которая предназначена для хранения ссылок на объекты которые будут получать события выбора цвета. Регистрация этих объектов получателей осуществляется при помощи метода `addOnColorPickListener()` (см. Листинг 3.14). Отмена регистрации объектов получателей осуществляется при помощи метода `removeOnColorPickListener()` (см. Листинг 3.14). И, наконец, оповещение объектов получателей о наступлении события смены выбранного цвета в виджете `MyColorPicker` осуществляется в методе `onTouch()`. В Листинге 3.14 этот фрагмент кода выделен жирным цветом. Теперь наш класс виджета выбора цвета `MyColorPicker` является логически завершенным и готовым для использования.

В методе `onCreate()` класса Активности зарегистрируем безымянный объект на получение событий смены цвета от виджета `MyColorPicker`. В Листинге 3.15 показан код регистрации и обработки события смены цвета (выделен жирным шрифтом). Внешний вид работы примера показан на Рис. 3.10.

Листинг 3.15. Код регистрации объекта получателя и обработка события смены цвета от виджета MyColorPicker

```
public class MainActivity extends AppCompatActivity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // ----- Toolbar -----
        final Toolbar toolBar =
            this.findViewById(R.id.toolbar);
        this setSupportActionBar(toolBar);
        toolBar.setTitle("app3 Module / Lesson_08");
        toolBar.setSubtitle("Create a custom view");
        toolBar.setTitleTextColor(Color.WHITE);
        toolBar.setSubtitleTextColor(Color.WHITE);

        // ----- Handling event of the selected color ---
        // ----- Обработка события смены выбранного цвета -
        MyColorPicker mcpOne =
            this.findViewById(R.id.mcpOne);

        mcpOne.addOnColorPickListener(
            new MyColorPicker.
            OnColorPickListener()
        {
            @Override
            public void onColorPick(int clr)
            {
                toolBar.setBackgroundColor(clr);
            }
        });
    }
}
```

Из Листинга 3.15 видим в обработчике события смены цвета выбранный в виджете `MyColorPicker` цвет назначается в качестве цвета фона для тулбара (см. Рис. 3.10).

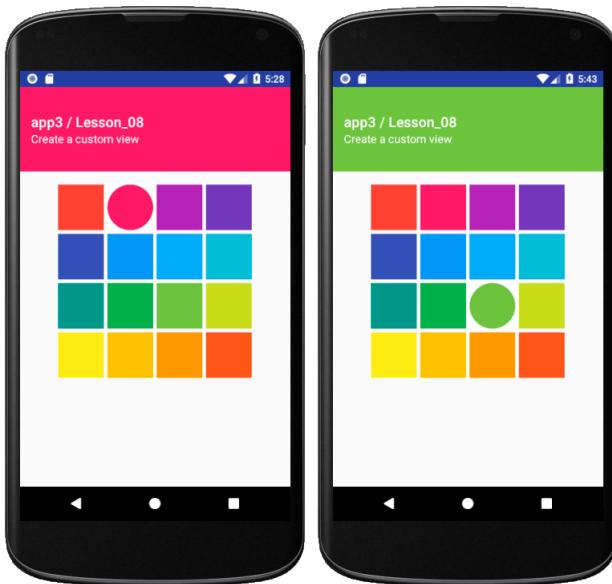


Рис. 3.10. Внешний вид работы примера с созданным кастомным виджетом `MyColorPicker`

Как видно из Рис. 3.10, созданный виджет `MyColorPicker` работает исправно. Однако, пока не будем считать рассмотренный пример полностью завершенным. Давайте попробуем убрать виджет `MyColorPicker` с Активности чтобы разместить его в Диалоговом окне `android.app.AlertDialog` (<https://developer.android.com/reference/android/app/AlertDialog>). Ведь виджет `MyColorPicker` логично использовать именно в Диалоговом окне. Для этого мы создадим модуль «app4» в проекте Android Studio, который прилагается к данному уроку.

Для содержимого Диалогового окна `android.app.AlertDialog` создадим файл ресурсов с макетом внешнего вида `/res/layout/dialog_view.xml`. Содержимое этого файла приведено в Листинге 3.16.

Листинг 3.16. Содержимое файла `/res/layout/dialog_view.xml` с макетом внешнего вида содержимого Диалогового окна `android.app.AlertDialog`

```
<?xml version="1.0" encoding="utf-8"?>
<itstep.com.myapp4.MyColorPicker
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:layout_width="280dp"
    android:layout_height="280dp"
    android:layout_margin="16dp"
    android:layout_gravity="center_horizontal"
    android:id="@+id/mcpOne"
/>
```

Как видно из Листинга 3.16, в качестве содержимого Диалогового окна выступает наш виджет `MyColorPicker`. В макете Активности разместим только кнопку `android.support.design.widget.FloatingActionButton` (<https://developer.android.com/reference/android/support/design/widget/FloatingActionButton>), по нажатию на которую будет появляться Диалоговое окно с виджетом выбора цвета `MyColorPicker`. Также, как и в предыдущем примере, при помощи виджета `MyColorPicker` мы будем менять цвет фона тулбара (идентификатор `R.id.toolbar`). Код xml верстки внешнего вида макета активности (файл ресурсов `/res/layout/activity_main.xml`) ввиду его понятности, не приводиться в Листингах этого урока. Внешний вид

Активности рассматриваемого примера можно увидеть на Рис. 3.11 слева.

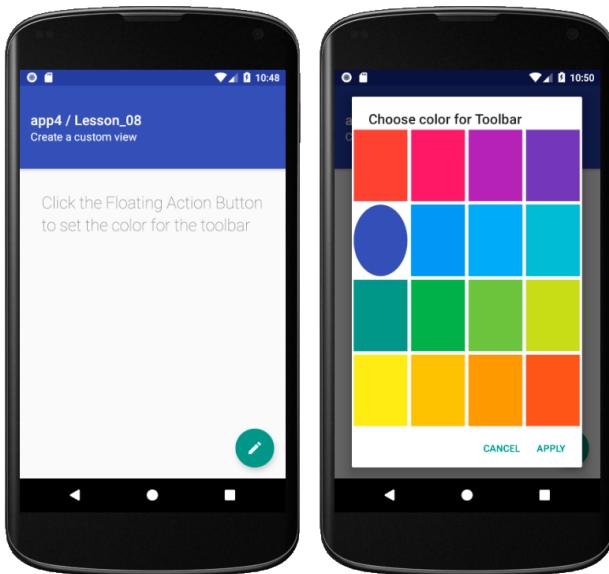


Рис. 3.11. Внешний вид Активности (слева) и Диалогового окна с виджетом выбора цвета MyColorPicker (справа)

Программный код использования Диалогового окна для виджета выбора цвета MyColorPicker также достаточно очевиден и поэтому приводится в Листинге 3.17 с сокращениями.

Листинг 3.17. программный код класса Активности MainActivity с использованием Диалогового окна с виджетом MyColorPicker

```
public class MainActivity extends AppCompatActivity
{
    // ----- Class members -----
    /**
     * 
```

```
* AlertDialog for the MyColorPicker widget
*
* AlertDialog для виджета выбора цвета MyColorPicker
*/
private AlertDialog dialog;

/**
 * The currently assigned toolbar background color
 *
 * Текущий назначенный цвет фона тулбара
 */
private int curToolbarClr;

/**
 * Reference to the toolbar for which
 * the background color is assigned
 *
 * Ссылка на тулбар для которого назначается
 * цвет фона
 */
private Toolbar toolBar;

// ----- Class methods -----
@Override
protected void onCreate(Bundle savedInstanceState)
{
    ...

/*
 * AlertDialog
 * -----
 */
// ----- Create AlertDialog.Builder -----
// ----- Создание объекта AlertDialog.Builder ---
    AlertDialog.Builder builder;
    ...
}
```

```
View dialogView =
        this.getLayoutInflater().inflate(
            R.layout.dialog_view, null,
            false);

// ----- Create View for AlertDialog -----
// ----- Создание виджета для содержимого
// ----- AlertDialog -----
final MyColorPicker mcpOne =
        dialogView.findViewById(R.
            id.mcpOne);
mcpOne.addOnColorPickListener(
        new MyColorPicker.
            OnColorPickListener()
{
    @Override
    public void onColorPick(int clr)
    {
        toolBar.setBackgroundColor(clr);
    }
});

// ----- Assign th Title and View to AlertDialog
// ----- Назначаем заголовок и виджет с внешним
// ----- видом Диалоговому окну -----
builder.setView(dialogView);
builder.setTitle("Choose color for Toolbar");

// ----- Set Negative and Positive Button
// ----- to AlertDialog -----
// ----- Обработчики событий на "Cancel"
// ----- и "Apply" -----
builder.setNegativeButton("Cancel",
        new DialogInterface.OnClickListener()
{
    @Override
```

```
public void onClick(DialogInterface dialogInterface, int i)
{
    toolBar.setBackgroundColor(
        MainActivity.this.curToolbarClr);
}
});

builder.setPositiveButton("Apply",
    new DialogInterface.OnClickListener()
{
    @Override
    public void onClick(DialogInterface dialogInterface, int i)
    {
        MainActivity.this.curToolbarClr =
            mcpOne.getColor();
    }
});
}

// ----- Build the AlertDialog -----
this.dialog = builder.create();
}

public void fabClick(View v)
{
    this.dialog.show();
}
}
```

Как видно из Листинга 3.17, в классе `MainActivity` объявлено поле `curToolbarColor`, которое предназначено для хранения текущего цвета фона тулбара (поле `toolbar`). Поясним, для чего необходимо это поле: Когда Диало-

говое окно `android.app.AlertDialog` (поле `dialog`) показано, и пользователь выбирает какой-либо цвет в виджете `MyColorPicker` (см. Рис. 3.11 справа), то выбранный цвет сразу становится цветом фона тулбара `toolbar`. Это происходит потому что виджету `MyColorPicker` (переменная `mcpOne`) назначен обработчик события смены выбранного цвета `MyColorPicker.OnColorPickListener` в котором при получении события смены цвета происходит назначение выбранного цвета тулбару `toolbar`. Однако, если пользователь нажмет в Диалоговом окне кнопку «Cancel» («Отмена»), то необходимо тулбару вернуть цвет фона, который был назначен ему до показа Диалогового окна. И это значение берется из поля `curToolbarColor`. Если же пользователь нажмет на кнопку «Apply» («Применить»), то полю `curToolbarColor` в обработчике нажатия на кнопку «Apply» присваивается новое значение цвета. Таким образом, поле `curToolbarColor` в любой момент времени хранит значение назначенного тулбару цвета фона.

В целом, использование нашего виджета `MyColorPicker` в Диалоговом окне `android.app.AlertDialog` показало полную готовность класса `MyColorPicker` к дальнейшему использованию в других проектах. За одним маленьким недостатком, который можно увидеть на Рис. 3.11 справа. Этот недостаток заключается в том, что виджет `MyColorPicker` может находиться в области, которая будет не квадратной, а прямоугольной. Из-за этого страдает внешний вид нашего виджета. Поэтому внесем в класс `MyColorPicker` (а точнее в метод `onDraw()`) соответствующие корректизы. Новый вариант метода `onDraw()` по-

казан в Листинге 3.18. Внесенные изменения показаны в Листинге 3.18 жирным шрифтом.

Листинг 3.18. Внесенные в метод onDraw() изменения для отображения рабочей области виджета MyColorPicker строго в виде квадрата с квадратными ячейками

```
@Override
public void onDraw(Canvas canvas)
{
    // ----- width of one cell -----
    // ----- ширина одной ячейки -----
    int w = this.getWidth() / 4;

    // ----- height of one cell -----
    // ----- высота одной ячейки -----
    int h = this.getHeight() / 4;

    // ----- Coordinates of the widget's working area
    // ----- offset -----
    // ----- Координаты смещения рабочей области
    // ----- виджета -----
    int x = 0;
    int y = 0;

    // ----- Calculating cell dimensions and work
    // ----- area offset coordinates -----
    // ----- Рассчет размеров ячейки и координат
    // ----- смещения рабочей области -----
    if (w > h)
    {
        w = h;
        x = (this.getWidth() - 4 * w) / 2;
    }
    else
    {
        h = w;
```

```
y = (this.getHeight() - 4 * h) / 2;  
}  
  
// ----- Fill the entire surface with  
// ----- a background white color -----  
// ----- Заполняем всю поверхность фоновым белым  
// ----- цветом -----  
    canvas.drawColor(Color.WHITE);  
  
// ----- Drawing available colors for selection  
// ----- in the form of a 4 by 4 grid-----  
// ----- Рисование доступных для выбора цветов  
// ----- в виде сетки 4 на 4 -----  
for (int i = 0; i < 4; i++)  
{  
    for (int j = 0; j < 4; j++)  
{  
  
    // ----- Color from the array of colors for  
    // ----- the current cell -----  
    // ----- Цвет из массива цветов для текущей клетки  
        this.P.setColor(colors[i * 4 + j]);  
  
    if (this.curColor == colors[i * 4 + j])  
{  
  
    // ----- The current selected color should be  
    // ----- displayed as a selected color.-----  
    // ----- Текущий выбранный цвет нужно отобразить  
    // ----- как выделенный -----  
        if (Build.VERSION.SDK_INT >= 21)  
            canvas.drawOval(x + 5 + j * w,  
                            y + 5 + i * h,  
                            x + (j + 1) * w - 5,  
                            y + (i + 1) * h - 5, this.P);  
        else  
        {
```

```
        canvas.drawRect(x + 15 + j * w,
                        y + 15 + i * h,
                        x + (j + 1) * w - 15,
                        y + (i + 1) * h - 15,
                        this.P);
    }
}
else
{
    canvas.drawRect(x + 5 + j * w,
                    y + 5 + i * h,
                    x + (j + 1) * w - 5,
                    y + (i + 1) * h - 5,
                    this.P);
}
}
}
```

Как видно из Листинга 3.18, ширина и высота каждой ячейки виджета **MyColorPicker** делается одинаковой. Причем, в качестве значения для стороны выбирается меньшее значение из значений текущей высоты и ширины ячейки. Таким образом, ячейка становится квадратной (и уменьшается в своих размерах или по ширине или по высоте). Но это еще не все. Из-за того, что ячейка становится меньше, должна уменьшиться и рабочая область виджета **MyColorPicker**. Под рабочей областью мы подразумеваем область виджета **MyColorPicker** в которой отображаются только ячейки с доступными цветами и в которую пользователь может нажимать для выбора понравившегося ему цвета. Уменьшая рабочую область,

мы позаботимся о том, чтобы эта область располагалась посередине той области, которая предоставлена виджету `MyColorPicker`. Таким образом, появляется необходимость в переменных `x` и `y`, которые содержат смещение начала рабочей области виджета `MyColorPicker` от его левого верхнего угла (то есть от начала его координат). Далее координаты смещения рабочей области `x` и `y` применяются при отрисовке каждой ячейки. Внешний вид полученного результата показан на Рис. 3.12.



Рис. 3.12. Внешний вид виджета `MyColorPicker` с внесенными изменениями из Листинга 3.18

Далее, поскольку изменились размеры и расположение рабочей области виджета `MyColorPicker`, то необходимо учесть этот факт и в методе `onTouch()`, который об-

работывает события нажатия пользователя и вычисляет ячейку, которая попадает в точку нажатия. Содержимое модифицированного метода `onTouch()` показано в Листинге 3.19.

Листинг 3.19. Содержимое модифицированного метода `onTouch()`, в котором при вычислении ячейки, в которую нажал пользователь, учитываются размеры и расположение рабочей области виджета `MyColorPicker`

```
@Override
public boolean onTouch (View v, MotionEvent event)
{
    // ----- Width of one cell -----
    // ----- Ширина одной ячейки -----
    int w = this.getWidth() / 4;

    // ----- Height of one cell -----
    // ----- Высота одной ячейки -----
    int h = this.getHeight() / 4;

    // ----- Coordinates of the widget's working area
    // ----- offset -----
    // ----- Координаты смещения рабочей области
    // ----- виджета -----
    int x = 0;
    int y = 0;

    // ----- Calculating cell dimensions and work
    // ----- area offset coordinates -----
    // ----- Рассчет размеров ячейки и координат
    // ----- смещения рабочей области -----
    if (w > h)
    {
        w = h;
        x = (this.getWidth() - 4 * w) / 2;
    }
}
```

```
else
{
    h = w;
    y = (this.getHeight() - 4 * h) / 2;
}

// ----- Coordinates of pressing relative to
// the beginning of the working area-----
// ----- Координаты нажатия относительно начала
// рабочей области -----
int pressX = (int) event.getX() - x;
int pressY = (int) event.getY() - y;

// ----- Verifying that the user made a click
// on the work area -----
// ----- Проверка, что пользователь сделал
// ----- нажатие в рабочей области -----
if (pressX < 0 || pressX > w * 4 ||
    pressY < 0 || pressY > h * 4) return true;

// ----- The row to which you clicked -----
// ----- Стока в которую кликнули -----
int row = pressY / h;

// ----- The column that was clicked -----
// ----- Столбец в который кликнули -----
int col = pressX / w;

// ----- Get the color that the user chose -----
// ----- Получаем цвет, который выбрал пользователь
this.curColor = colors[row * 4 + col];

// ----- System! As soon as possible, call
// ----- the onDraw () method!
// ----- Система! Как можно быстрее вызови метод
// ----- onDraw()!
// ----- -----
this.invalidate();

// ----- Alerting all listeners about the color
// ----- change -----
```

```
// ----- Оповещение всех слушателей о смене цвета
for (int i = 0; i < this.listeners.size(); i++)
{
    this.listeners.get(i).onColorPick(this.curColor);
}
return true;
}
```

Как видно из Листинга 3.19, в методе `onTouch()` при определении ячейки, в которую нажал пользователь, учитываются размеры и расположение рабочей области виджета `MyColorPicker`. И если пользователь нажал на виджет `MyColorPicker` и не попал в рабочую область, то никаких событий не происходит.

Теперь в нашем распоряжении находится полностью готовый к использованию в разных приложениях виджет выбора цвета (класс `MyColorPicker`).

Исходный код примеров, рассмотренных в данном разделе, находится в модулях «app3» и «app4» в проекте Android Studio, который прилагается к данному уроку.

Окончательная версия класса `MyColorPicker` находится в модуле «app4».

3.3. Пример приложения с динамически обновляющейся графикой с использованием метода `invalidate()` и события `onDraw()`

В данном разделе мы рассмотрим пример создания приложения с динамически обновляющейся 2D графикой с использованием метода `invalidate()` класса `android.`

`view.View`. Сразу же отметим, что использование метода `invalidate()` для приложений с обновляющейся графикой имеет свои плюсы и минусы. К плюсам можно отнести легкость и быстроту программирования. К минусам относится то, что перерисовка не происходит так быстро, как этого требуют некоторые современные игры с мощной графикой. Разработчикам Android приложений, кроме инструмента `invalidate()`, предлагаются и другие инструменты для создания приложений с обновляющейся графикой. В частности, таким инструментом является использование класса `android.view.SurfaceView`. Использование этого класса дает возможность создавать приложения с быстро обновляющейся графикой и будет рассмотрено в следующем уроке данного курса.

Что касается создания графических приложений, в которых графические изменения не происходят с высокой скоростью, то использование инструмента `invalidate()` для таких приложений является хорошим решением.

Для примера данного раздела создан модуль «app5» в проекте Android Studio, который прилагается к данному уроку. Суть примера состоит в следующем: в виджете будет происходить бесконечное движение шарика, который при «столкновении» с границами виджета будет менять направление своего движения. Внешний вид виджета (занимает всю область Активности) показан на Рис. 3.13.

Класс виджета будет называться `MyView`. Этот класс будет, как и в предыдущих разделах, наследником от класса `android.view.View`. Также, этот класс будет реализовывать интерфейс `java.lang.Runnable`. То есть, экземпляр этого

класса будет вторичным потоком. Во вторичном потоке будет происходить изменение координат шарика с течением времени и вызов метода `invalidate()` когда возникнет необходимость перерисовать виджет `MyView`. Конкретнее, работа вторичного потока сводится к выполнению следующих действий в бесконечном цикле: изменение координат шарика, сообщение о необходимости перерисовать виджет (вызов `invalidate()`), временная задержка (вызов `Thread.sleep()`), изменение координат шарика, сообщение о необходимости перерисовать виджет (вызов `invalidate()`), временная задержка (вызов `Thread.sleep()`) и так далее. Программный код класса `MyView` приведен в Листинге 3.20.

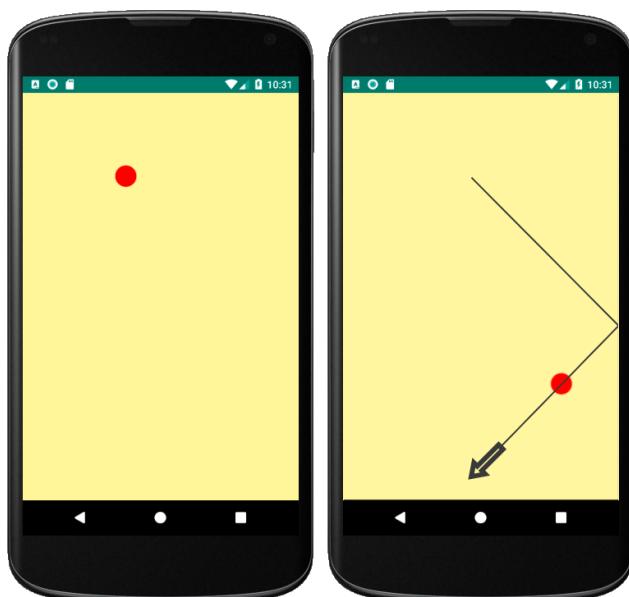


Рис. 3.13. Внешний вид примера который рассматривается в данном разделе. Справа показана траектория движения шарика после столкновения с границей виджета

Листинг 3.20. Программный код класса MyView рассмотриваемого примера

```
class MyView extends View implements Runnable
{
    // ----- Class members -----
    /**
     * An object with the drawing settings that is
     * used in the onDraw () method
     *
     * Объект с настройками рисования, который
     * используется в методе onDraw()
     */
    private Paint P;

    /**
     * Object with the coordinates of the ball
     *
     * Объект с координатами шарика
     */
    private RectF R;

    /**
     * Radius of the ball
     *
     * Радиус шарика
     */
    private final int radius = 30;

    /**
     * The distance to which the ball moves on each
     * time interval
     *
     * Расстояние, на которое перемещается шарик
     * на каждом интервале времени
     */
    private final int distance = 5;
```

```
/**  
 * The X-coordinate of the center of the ball  
 *  
 * X-координата центра шарика  
 */  
private int x = radius;  
  
/**  
 * The Y-coordinate of the center of the ball  
 *  
 * Y-координата центра шарика  
 */  
private int y = radius;  
  
/**  
 * The direction of movement of the ball along  
 * the X axis  
 *  
 * Направление движения шарика по оси X  
 */  
private boolean isForwardX = true;  
  
/**  
 * The direction of movement of the ball along  
 * the Y axis  
 *  
 * Направление движения шарика по оси Y  
 */  
private boolean isForwardY = true;  
  
/**  
 * The secondary thread working flag.  
 * If false, the thread should stop working.  
 *  
 * Признак работы вторичного потока.  
 * Если false, то поток должен прекратить свою работу.  
 */
```

```
private boolean isRun = true;

/**
 * The secondary thread stopping flag.
 * If false, then the thread should work,
 * otherwise – stand.
 *
 * Признак приостановки вторичного потока.
 * Если false то поток должен работать, иначе –
 * стоять.
 */
private boolean isStopped = true;

public MyView(Context context)
{
    super(context);
}

/**
 * Required to override the constructor!
 *
 * Обязательный для переопределения конструктор!
 */
public MyView(Context context, AttributeSet attrs)
{
    super(context, attrs);
}

/**
 * Required to override the constructor!
 *
 * Обязательный для переопределения конструктор!
 */
public MyView(Context context, AttributeSet
              attrs, int defStyleAttr)
{
```

```
super(context, attrs, defStyleAttr);  
}  
  
// ----- Anonymous method -----  
// ----- Безымянный метод -----  
{  
    this.P = new Paint();  
    this.P.setAntiAlias(true);  
    this.P.setColor(Color.RED);  
    this.R = new RectF();  
  
    // ----- Starting the secondary thread -----  
    // ----- Запуск вторичного потока -----  
    Thread T = new Thread(this);  
    T.start();  
}  
  
@Override  
public void run()  
{  
    try  
    {  
        while (this.isRun)  
        {  
            // ----- Check for the suspension of the thread  
            // ----- Проверка на приостановку работы потока  
            this.checkStopped();  
  
            // ----- Calculating the distance to which  
            // ----- the ball should move -----  
            // ----- Вычисление расстояния на которое должен  
            // ----- переместиться шарик -----  
            int w = this.getWidth();  
            int h = this.getHeight();  
            this.x += this.distance *  
                ((this.isForwardX)?1:-1);  
        }  
    }  
}
```

```
        this.y += this.distance *
                ((this.isForwardY)?1:-1);

// ----- Check to achieve widget boundaries,
// ----- i.e. to "collision" -----
// ----- Проверка на достижение границ виджета,
// ----- т.е. на "столкновение" -----
        if (this.isForwardX && this.x +
            this.radius >= w ||

            !this.isForwardX && this.x -
            this.radius <= 0)
        {
            this.isForwardX = !this.isForwardX;
        }

        if (this.isForwardY && this.y +
            this.radius >= h ||

            !this.isForwardY && this.y -
            this.radius <= 0)
        {
            this.isForwardY = !this.isForwardY;
        }

// ----- Messages to the widget about the need
// ----- to redraw its appearance -----
// ----- Сообщаем виджету о необходимости
// ----- перерисовки своего внешнего вида-----
        this.postInvalidate();

// ----- Delay -----
// ----- Задержка -----
        Thread.sleep(30);
    }
}

catch (InterruptedException ie)
{
```

```
        Log.d("=====", "Thread 'MyView' Interrupted");
    }
}

@Override
public void onDraw(Canvas canvas)
{
// ----- Fill the widget area with the color
// ----- 0xFFFF59D -----
// ----- Заполнение всей области виджета цветом
// ----- 0xFFFF59D -----
    canvas.drawColor(Color.rgb(0xFF, 0xF5, 0x9D));

// ----- Coordinates of the ball -----
// ----- Координаты шарика -----
    this.R.left = this.x - this.radius;
    this.R.top = this.y - this.radius;
    this.R.right = this.x + this.radius;
    this.R.bottom = this.y + this.radius;

// ----- Drawing a ball in current coordinates
// ----- Рисование шарика в текущих координатах
    canvas.drawOval(this.R, this.P);
}

/**
 * The method is designed to call to stop
 * the secondary thread when the widget is
 * destroyed.
 *
 * Метод предназначен для вызова чтобы остановить
 * работу вторичного потока при уничтожении виджета.
 */
public void destroyView()
{
    this.isRun = false;
```

```
        this.startMoving();
    }

/***
 * The method is designed to suspend the thread
 * Метод предназначен для приостановки работы потока
 */
public synchronized void stopMoving()
{
    this.isStopped = true;
}

/***
 * The method is designed to resume the previously
 * suspended thread
 * Метод предназначен для возобновления работы
 * ранее приостановленного потока
 */
public synchronized void startMoving()
{
    this.isStopped = false;
    this.notify();
}

/**
 * The method checks the need to suspend the thread
 * Метод проверяет необходимость приостановить
 * работу потока
 *
 * @throws InterruptedException
 */
private synchronized void checkStopped()
                    throws InterruptedException
{
    while (this.isStopped)
    {
```

```
        this.wait();
    }
    if (!this.isRun) throw new InterruptedException();
}
}
```

Давайте рассмотрим класс `MyView` из Листинга 3.20. Этот класс является виджетом (то есть является наследником класса `android.view.View`), а также реализует интерфейс `java.lang.Runnable`, то есть является классом вторичного потока. В качестве виджета, объект `MyView` предназначен для отображения на Активности, а вторичный поток предоставляет возможность объекту `MyView` динамически менять свое состояние, и, как следствие, свой внешний вид. В классе `MyView` объявлены поля `x` и `y`, которые содержат координаты шарика, объявлено поле `radius`, которое задает размер шарика в виде радиуса, объявлено поле `distance`, которое задает расстояние, на которое будет перемещаться шарик за каждый интервал времени, который равен 30 миллисекундам. Также в классе `MyView` объявлены поля `isForwardX` и `isForwardY`, которые содержат информацию о направлении движения шарика по оси X и Y соответственно. Если `true` — то шарик летит по направлению оси, в противном случае, шарик летит в противоположном направлении оси направлении. В основе главной функции вторичного потока (метод `run()`) лежит цикл, в котором происходит вычисление новых координат шарика и проверка, не «столкнулся» ли шарик с границами виджета. Если шарик столкнулся с верхней

или нижней границами виджета, то меняется направление движения шарика по оси Y, то есть поле `isForwardY` принимает значение `!isForwardY`. Если шарик столкнулся с левой или правой границами виджета, то меняется направление движения шарика по оси X, то есть изменяется поле `isForwardX`. После того, как новые координаты шарика рассчитаны, происходит вызов метода `postInvalidate()`, чтобы сообщить виджету `MyView` о необходимости перерисовки своего внешнего вида. После вызова метода `postInvalidate()`, вторичный поток приостанавливает свою работу на 30 миллисекунд при помощи вызова метода `Thread.sleep()`. После выхода из метода `Thread.sleep()` поток начинает новую итерацию цикла своей работы.

Хотим обратить ваше внимание на метод `postInvalidate()` класса `android.view.View`. Этот метод является аналогом метода `invalidate()`. Но, в отличии от метода `invalidate()`, метод `postInvalidate()` предназначен для вызова из вторичных потоков. Попытка явного вызова метода `invalidate()` из вторичного потока привела бы к исключительной ситуации. Во избежание возникновения исключительной ситуации, метод `invalidate()` можно было бы вызвать так, как показано в Листинге 3.21 (подразумевается, что этот код находится в методе `run()`).

Листинг 3.21. Вызов метода `invalidate()` из вторичного потока

```
this.post(
    new Runnable()
    {
        @Override
```

```
    public void run()
    {
        MyView.this.invalidate();
    }
});
```

Как видно из Листинга 3.21, вызвать метод `postInvalidate()` из вторичного потока оказывается куда более простым, чем вызов метода `invalidate()`. А результат один и тот же.

Также в классе `MyView` есть методы, предназначенные для остановки и возобновления работы потока:

- метод `void stopMoving()` — предназначен для приостановки работы потока `MyView`. Этот метод необходимо вызывать когда Активность получает событие `onPause()` (см. Листинг 3.22).
- метод `void startMoving()` — предназначен для возобновления работы ранее приостановленного потока `MyView`. Метод необходимо вызывать когда Активность получает событие `onResume()` (см. Листинг 3.22).
- метод `void destroyView()` — предназначен для завершения работы вторичного потока `MyView`. Этот метод необходимо вызывать когда Активность получает событие `onDestroy()` (см. Листинг 3.22).

Перечисленные методы созданы в классе `MyView` в соответствии с темой «Многопоточность в Android приложениях», которая рассматривалась нами в прошлых уроках. Если у вас возникли вопросы по этим методам, то рекомендуем еще раз рассмотреть эту тему.

И, наконец, приведем код класса Активности **MainActivity**, в котором происходит создание, приостановка и уничтожение объекта **MyView**. Код класса **MainActivity** приведен в Листинге 3.22.

Листинг 3.22. Класс **MainActivity** рассматриваемого примера

```
public class MainActivity extends AppCompatActivity
{
    // ----- Class members -----
    private MyView MV;

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

        MV = new MyView(this);
        this.setContentView(MV);
    }

    @Override
    protected void onResume()
    {
        super.onResume();
        this.MV.startMoving();
    }

    @Override
    protected void onPause()
    {
        super.onPause();
        this.MV.stopMoving();
    }
}
```

```
@Override  
public void onDestroy()  
{  
    super.onDestroy();  
    this.MV.destroyView();  
}  
}
```

Как видно из Листинга 3.22, виджет `MyView` назначается в качестве содержимого всей Активности (хотя это и не обязательно — виджет `MyView` может занимать и часть Активности). Также, в методах обработчиках событий жизненного цикла Активности (`onResume()`, `onPause()`, `onDestroy()`) вызываются методы класса `MyView` для управления работы вторичным потоком (`startMoving()`, `stopMoving()`, `destroyView()` соответственно).

Запустите на эмуляторе или устройстве пример из данного раздела. Мы уверены, что вы почувствуете, что создавать игровые приложения для Android устройств не так уж и сложно.

Исходный код рассматриваемого в данном разделе примера находится в модуле «app5» в проекте Android Studio, который прилагается к данному уроку.

4. Основы работы с датчиками устройства на примере датчика акселерометра

Наверняка вам уже известно, что в Android устройствах находятся всевозможные датчики. Их еще называют «сенсоры» (от англ. слова «sensor»). В каждом устройстве может быть свой набор датчиков. В большинстве устройств есть датчики акселерометра и гироскопа. В целом, датчики в Android устройствах можно разделить на три вида: датчики движения, датчики положения и датчики окружающей среды. Любой датчик представляется классом `android.hardware.Sensor` (<https://developer.android.com/reference/android/hardware/Sensor>). В этом же классе объявлены константы типов датчиков, которые могут присутствовать в устройстве. Вот некоторые из них:

- **TYPE_ACCELEROMETER** — датчик акселерометра. Измеряет ускорение в пространстве по осям X, Y, Z.
- **TYPE_AMBIENT_TEMPERATURE** — датчик температуры окружающей среды.
- **TYPE_GRAVITY** — датчик силы тяжести.
- **TYPE_GYROSCOPE** — датчик гироскопа. Позволяет получить текущее положение устройства в пространстве в градусах по трём осям
- **TYPE_LIGHT** — датчик освещения. Измеряет степень освещенности. Этот тип датчика обычно используется для динамического изменения яркости экрана.

- **TYPE_LINEAR_ACCELERATION** — датчик линейного ускорения. Это виртуальный датчик, использующий показания акселерометра. Возвращает показатели ускорения без учёта силы тяжести.
- **TYPE_MAGNETIC_FIELD** — датчик магнитного поля.
- **TYPE_MOTION_DETECT** — датчик обнаружения движения.
- **TYPE_PRESSURE** — датчик давления (барометр). Возвращает текущее давление в миллибарах.
- **TYPE_PROXIMITY** — датчик приближения (расстояния), который определяет расстояние между устройством и целевым объектом в сантиметрах.
- **TYPE_RELATIVE_HUMIDITY** — датчик относительной влажности в виде процентного значения.
- **TYPE_STEP_COUNTER** — датчик счетчика шагов.

Для работы с датчиком необходимо получить у системы (при помощи объекта `android.hardware.SensorManager`) ссылку на соответствующий объект `android.hardware.Sensor`, представляющий необходимый датчик. Затем в методе жизненного цикла `onResume()` зарегистрироваться на получение событий от этого датчика, а в методе `onPause()` отменить регистрацию. При работе с датчиками необходимо помнить следующее:

1. Данные от датчиков приходят неравномерно.
2. Показания датчиков бывают неровными, поэтому необходимо использовать среднее значение для того чтобы приложение правильно реагировало на датчик.

Это среднее значение зачастую приходится подбирать методом проб при тестировании приложения.

На этом мы завершаем вводное знакомство с датчиками Android устройств. Как уже упоминалось выше, для работы с датчиками используется объект android.hardware.SensorManager (<https://developer.android.com/reference/android/hardware/SensorManager>). Давайте перейдем к рассмотрению этого объекта.

4.1. Класс android.hardware.SensorManager

Объект android.hardware.SensorManager (<https://developer.android.com/reference/android/hardware/SensorManager>) предназначен для получения доступа к датчикам устройства. Также при помощи этого объекта можно получить информацию, какие датчики есть в наличии у устройства.

Иерархия классов для класса android.hardware.SensorManager имеет следующий вид:

```
java.lang.Object
  |
  +--- android.hardware.SensorManager
```

Чтобы получить ссылку на объект android.hardware.SensorManager необходимо воспользоваться методом класса android.content.Context:

```
Object getSystemService (String name);
```

в который необходимо передать значение Context.SENSOR_SERVICE, как это показано в Листинге 4.1.

Листинг 4.1. Получение ссылки на объект android.hardware.SensorManager

```
public class MainActivity extends AppCompatActivity
{
    // ----- Class members -----
    private SensorManager SM;

    // ----- Class methods -----
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // ----- Get the SensorManager -----
        this.SM = (SensorManager) this.
            getSystemService(Context.
                SENSOR_SERVICE);
    }
}
```

Давайте познакомимся с некоторыми методами класса `android.hardware.SensorManager`:

- `Sensor getDefaultSensor(int type)` — Метод возвращает датчик (объект `android.hardware.Sensor`) по умолчанию для заданного в параметре `type` типа. В качестве типа датчика указывается какое-либо значение из констант, объявленных в классе `android.hardware.Sensor` (<https://developer.android.com/reference/android/hardware/Sensor>). Например `Sensor.TYPE_ACCELEROMETER`, `Sensor.TYPE_AMBIENT_TEMPERATURE`, и так далее. О некоторых из этих констант уже рассказывалось выше. Также, в официальной документации говорится, что метод `getDefaultSensor()` может вернуть датчик, кото-

рый может быть составным датчиком, и данные этого датчика могут быть усреднены или отфильтрованы. Поэтому, если необходимо получить доступ к оригинальным датчикам, то необходимо использовать метод `getSensorList()`. Метод `getDefaultSensor()` может вернуть `null`, если запрашиваемого датчика в устройстве нет или у приложения отсутствуют необходимые привилегии на работу с этим датчиком.

- `List<Sensor> getSensorList(int type)` — Метод получает список доступных датчиков указанного в параметре `type` типа. Для получения списка всех доступных датчиков в параметре `type` необходимо передать значение `Sensor.TYPE_ALL` (<https://developer.android.com/reference/android/hardware/Sensor>).
- `boolean registerListener(SensorEventListener listener, Sensor sensor, int samplingPeriodUs)` — Метод регистрирует обработчик события `android.hardware.SensorEvent` (<https://developer.android.com/reference/android/hardware/SensorEventListener.html>), ссылка на который передается в параметре `listener`, для датчика `sensor` на указанной в параметре `samplingPeriodUs` частоте дискретизации. Об интерфейсе `android.hardware.SensorEvent` расскажем чуть ниже. Параметр `samplingPeriodUs` задает частоту получения данных от датчика (частоту дискретизации). Значения этого параметра всего лишь пожелание операционной системе о намерении получать данные от датчика с указанной частотой. Данные от датчика могут приходить быстрее или медленнее, чем указано в параметре `samplingPeriodUs`.

Обычно данные от датчиков приходят быстрее. Для параметра `samplingPeriodUs` объявлены следующие константы в классе `android.hardware.SensorManager`: `SENSOR_DELAY_NORMAL` (частота по умолчанию, подходящая для отслеживания изменений ориентации устройства), `SENSOR_DELAY_UI` (частота, подходящая для отслеживания изменений, характерных для пользовательского интерфейса), `SENSOR_DELAY_GAME` (частота, подходящая для отслеживания изменений, характерных для игровых приложений), `SENSOR_DELAY_FASTEST` (получать данные от датчика как можно быстрее).

- `void unregisterListener(SensorEventListener listener, Sensor sensor)` — Метод отменяет регистрацию обработчика событий `listener` для датчика `sensor`.
- `void unregisterListener(SensorEventListener listener)` — Метод отменяет регистрации обработчика событий `listener` для всех датчиков.

В интерфейсе `android.hardware.SensorEventListener` (<https://developer.android.com/reference/android/hardware/SensorEventListener>) объявлены следующие методы:

- `void onAccuracyChanged(Sensor sensor, int accuracy)` — Метод вызывается, когда точность датчика `sensor` изменилась. Параметр `sensor` содержит ссылку на датчик (на объект `android.hardware.Sensor`) на получение данных от которого зарегистрирован объект, для которого вызывается метод `onAccuracyChanged()`. Параметр `accuracy` содержит информацию о новой точности этого датчика в виде одного из значений констант `SensorManager`.

SENSOR_STATUS_XXXX (<https://developer.android.com/reference/android/hardware/SensorManager.html>).

- void onSensorChanged(SensorEvent event) — Метод вызывается когда доступны новые данные от датчика. Допускается ситуация, когда новые значения данных датчика могут быть такими же, как и предыдущие значения, но для текущего времени. Параметр event содержит информацию о событии от датчика.

Информация о данных датчика приходит в объекте события android.hardware.SensorEvent (<https://developer.android.com/reference/android/hardware/SensorEvent>).

В этом объекте объявлены следующие поля:

- int accuracy — Точность в виде одного из значений констант SensorManagerSENSOR_STATUS_XXXX (<https://developer.android.com/reference/android/hardware/SensorManager.html>).
- Sensor sensor — Ссылка на датчик (на объект android.hardware.Sensor) который является источником данного события.
- long timestamp — Время наступления события в наносекундах.
- final float[] values — Массив с данными датчика. Количество значений в массиве и содержимое этого массива зависит от типа датчика, от которого пришло событие. Поэтому, информацию о содержании массива необходимо искать в справочной документации API. Информация о массиве values для некоторых датчиков приводится на странице <https://developer.android.com/reference/android/hardware/SensorEvent>.

Что касается датчика акселерометра, то в массиве `values` объекта `android.hardware.SensorEvent` приходят следующие данные:

- `values[0]` — ускорение по оси X.
- `values[1]` — ускорение по оси Y.
- `values[2]` — ускорение по оси Z.

Направление осей координат в Android устройствах показано на Рис. 4.1.

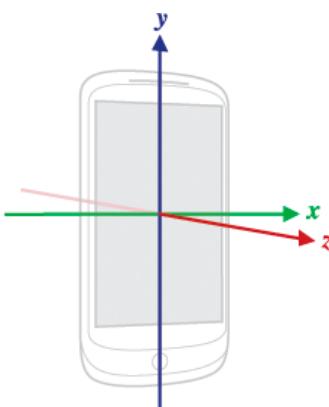


Рис. 4.1. Направление осей координат
в Android устройствах

Теперь, когда мы получили основные знания о классе `android.hardware.SensorManager` и других классах из пакета `android.hardware.*`, давайте рассмотрим, как зарегистрировать объект обработчик события на получение событий от датчика, на примере датчика акселерометра. В Листинге 4.2 показан программный код, демонстрирующий регистрацию, отмену регистрации на получение событий от датчика акселерометра, а также получение данных от этого датчика.

Листинг 4.2. Регистрация и отмена регистрации обработчика событий получения данных от датчика акселерометра, а также получение данных от датчика акселерометра

```
public class MainActivity extends AppCompatActivity
    implements SensorEventListener
{
    // ----- Class members -----
    private SensorManager SM;
    private Sensor accelerometerSensor;

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

        // ----- Get the SensorManager -----
        this.SM = (SensorManager)
            getSystemService(Context.
                SENSOR_SERVICE);

        // ----- Get the Accelerometer Sensor -----
        this.accelerometerSensor =
            this.SM.getDefaultSensor(Sensor.
                TYPE_ACCELEROMETER);
    }

    @Override
    public void onAccuracyChanged(Sensor sensor,
                                  int accuracy)
    {

    }

    @Override
    public void onSensorChanged(SensorEvent event)
    {
```

```
    float [] values = event.values;
    switch(event.sensor.getType())
    {
        case Sensor.TYPE_ACCELEROMETER:
// ----- Getting data from sensor here -----
        float gX = values[0];
        float gY = values[1];
        float gZ = values[2];
        break;
    }
}

@Override
protected void onResume()
{
    super.onResume();
    this.SM.registerListener(this,
        this.accelerometerSensor,
        SensorManager.SENSOR_DELAY_GAME);
}

@Override
protected void onPause()
{
    this.SM.unregisterListener(this);
    super.onPause();
}
}
```

Как видно из Листинга 4.2, класс `MainActivity` реализует интерфейс `android.hardware.SensorEventListener` чтобы получать события от датчика акселерометра. Регистрация на получение событий от датчика акселерометра осуществляется в методе `onResume()` жизненного цикла Активности (см. Листинг 4.2). Отмена регистрации

осуществляется в методе `onPause()`. Получение данных (в Листинге 4.2 этот фрагмент кода выделен жирным шрифтом) осуществляется в методе `onSensorChanged()` класса `MainActivity`.

Перейдем теперь к рассмотрению конкретных примеров на получение данных от датчика акселерометра.

4.2. Получение данных с датчика акселерометра

В данном разделе мы рассмотрим два практических примера: пример, в котором будет получен список всех доступных датчиков устройства, и пример, в котором будет происходить получение данных с датчика акселерометра. Для этих примеров созданы модули «app9» и «app10» в проекте Android Studio, который прилагается к данному уроку.

Итак, первым примером (модуль «app9») будет пример получения списка всех доступных датчиков устройства. Список всех доступных датчиков будет выводиться в текстовое поле (виджет `android.widget.TextView`) с идентификатором `R.id.tvSensorList`, которое размещено на Активности. Содержимое файла ресурсов `/res/layout/activity_main.xml` с версткой макета внешнего вида Активности показано в Листинге 4.3.

Листинг 4.3. Макет Активности для примера получения списка доступных датчиков устройства

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

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="@color/colorPrimary"
        />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textSize="20sp"
            android:layout_margin="8dp"
            android:id="@+id/tvSensorList"
            />
    </ScrollView>
</LinearLayout>
```

Как видно из Листинг 4.3, текстовое поле R.id.tvSensorList помещено внутрь контейнера android.widget ScrollView с целью обеспечения скроллинга для случая, если список датчиков будет достаточно длинным и не будет помещаться на экране устройства.

Получение списка доступных датчиков и отображение этого списка в текстовом поле R.id.tvSensorList происходит в методе `onCreate()` класса Активности `MainActivity`. Этот программный код показан в Листинге 4.4.

Листинг 4.4. Получение списка доступных датчиков устройства

```
public class MainActivity extends AppCompatActivity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // ----- Toolbar -----
        Toolbar toolBar =
            this.findViewById(R.id.toolbar);
        this.setSupportActionBar(toolBar);
        toolBar.setTitle("app9 / Lesson_08");
        toolBar.setSubtitle("Sensor List");
        toolBar.setTitleTextColor(Color.WHITE);
        toolBar.setSubtitleTextColor(Color.WHITE);

        // ----- Get the SensorManager -----
        // ----- Получение ссылки на android.hardware.
        // ----- SensorManager -----
        SensorManager SM = (SensorManager)
            getSystemService(Context.
                SENSOR_SERVICE);

        // ----- Getting a list of sensors -----
        // ----- Получение списка датчиков -----
        List<Sensor> sensors = SM.getSensorList(
            Sensor.TYPE_ALL);
        StringBuilder SB = new StringBuilder();
        for (Sensor S : sensors)
        {
            SB.append(S.getName());
            SB.append("\n");
        }
    }
}
```

```
// ----- Displaying the list of sensors in
// ----- the text field tvSensorList -----
// ----- Отображение списка датчиков в текстовом
// ----- поле tvSensorList -----
    ((TextView)this.findViewById(R.
        id.tvSensorList)).
    setText(SB.toString());
}
}
```

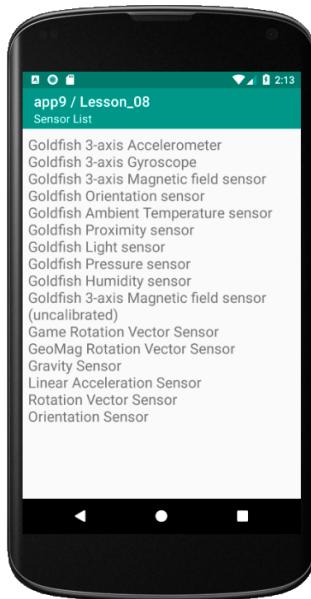


Рис. 4.2. Внешний вид работы примера из Листингов 4.3 и 4.4 в эмуляторе

Как видно из Листинга 4.4, список доступных датчиков получается при помощи вызова метода `getSensorList()` (в Листинге 4.4 этот фрагмент кода выделен жирным шрифтом). Далее происходит перебор полученного спи-

ска и формирование строки для отображения в текстовом поле `R.id.tvSensorList`. Внешний вид работы примера из Листингов 4.3 и 4.4 в эмуляторе показан на Рис. 4.2.

Как видно из Рис. 4.2, на стандартном эмуляторе доступно значительное количество датчиков. Для дополнения к примеру, приложение из модуля «app9» было запущено на реальном устройстве «Samsung J7 2017». Внешний вид работы приложения «app9» на реальном устройстве показан на Рис. 4.3.

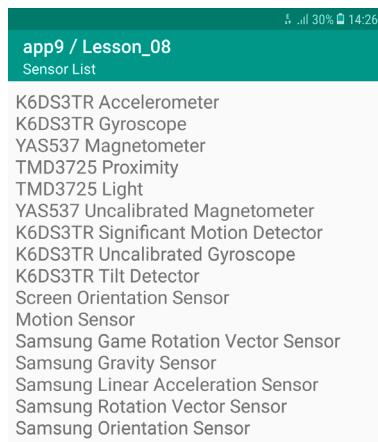


Рис. 4.3. Внешний вид работы примера из Листингов 4.3 и 4.4 на реальном устройстве

Как уже упоминалось выше и как видно на Рис. 4.3, количество и типы датчиков на разных устройствах разное.

Исходный код примера из Листингов 4.3 и 4.4 находится в модуле «app9» проекта Android Studio, который прилагается к данному уроку.

А мы перейдем к следующему примеру, в котором реализуем функциональность получения данных с датчика

акселерометра. Для этой цели создан модуль «app10» в проекте Android Studio, который прилагается к данному уроку.

Как уже говорилось выше, данные от датчика акселерометра приходят в виде значения ускорения по осям координат X, Y, Z. Поэтому на Активности разместим три текстовых поля `android.widget.TextView` с идентификаторами `R.id.tvValueX`, `R.id.tvValueY`, `R.id.tvValueZ`. Как вы видите, в название идентификатора включена информация о том, какое значение датчика акселерометра будет отображать соответствующее текстовое поле. Содержимое файла ресурсов `/res/layout/activity_main.xml` с макетом внешнего вида Активности в Листингах данного урока приводить не будем в силу своей понятности. В Листинге 4.2 приводился пример получения ссылки на объект `android.hardware.Sensor`, который представляет датчик акселерометра. Также, в Листинге 4.2 приводился пример регистрации и отмены регистрации на получение событий от датчика акселерометра. Поэтому пример из модуля «app10» дополняется функциональностью получения данных от датчика акселерометра и отображению этих данных в текстовых полях `R.id.tvValueX`, `R.id.tvValueY`, `R.id.tvValueZ`. Программный код класса Активности `MainActivity` из модуля «app10» приведен в Листинг 4.5.

Листинг 4.5. Класс `MainActivity` из модуля «app10»

```
public class MainActivity extends AppCompatActivity
    implements SensorEventListener
{
    // ----- Class members -----
    private SensorManager SM;
```

```
private Sensor accelerometerSensor;

private TextView tvValueX;
private TextView tvValueY;
private TextView tvValueZ;

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

// ----- Toolbar -----
Toolbar toolBar = this.findViewById(R.
        id.toolbar);
this.setSupportActionBar(toolBar);
toolBar.setTitle("app10 / Lesson_08");
toolBar.setSubtitle("Accelerometer Data");
toolBar.setTitleTextColor(Color.WHITE);
toolBar.setSubtitleTextColor(Color.WHITE);

// ----- Initializing the class members -----
this.tvValueX = this.findViewById(R.
        id.tvValueX);
this.tvValueY = this.findViewById(R.
        id.tvValueY);
this.tvValueZ = this.findViewById(R.
        id.tvValueZ);

// ----- Get the SensorManager -----
this.SM = (SensorManager)
        getSystemService(Context.
        SENSOR_SERVICE);

// ----- Get the Accelerometer Sensor -----
```

```
        this.accelerometerSensor =
            this.SM.getDefaultSensor(Sensor.
                TYPE_ACCELEROMETER);
    }

@Override
protected void onResume()
{
    super.onResume();
    this.SM.registerListener(this,
        this.accelerometerSensor,
        SensorManager.SENSOR_DELAY_GAME);
}

@Override
protected void onPause()
{
    this.SM.unregisterListener(this);
    super.onPause();
}

@Override
public void onAccuracyChanged(Sensor sensor,
                               int accuracy)
{
}

@Override
public void onSensorChanged(SensorEvent event)
{
    float [] values = event.values;
    switch(event.sensor.getType())
    {
        case Sensor.TYPE_ACCELEROMETER:
        {
            this.tvValueX.setText(String.
                format("%1.3f", values[0]));
        }
    }
}
```

```
        this.tvValueY.setText(String.  
            format("%1.3f", values[1]));  
    this.tvValueZ.setText(String.  
            format("%1.3f", values[2]));  
    }  
    break;  
}  
}  
}
```

Внешний вид приложения из Листинга 4.5 в эмуляторе изображен на Рис. 4.4.

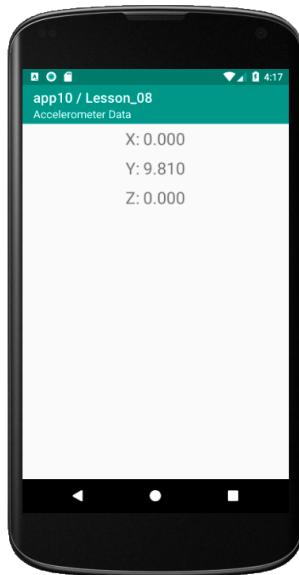


Рис. 4.4. Внешний вид работы примера из Листинга 4.5 в эмуляторе

Как видно из Рис. 4.4, данные от акселерометра поступают, но поскольку эмулируемое устройство находится

в состоянии покоя, то данные от датчика приходят постоянные. Для того, чтобы с целью тестирования поворачивать устройство и увидеть изменения в поступающих от акселерометра данных, необходимо воспользоваться специальным диалоговым окном «Extended Controls». Для того чтобы вызвать это диалоговое окно, необходимо на эмуляторе нажать на кнопку «...». На Рис. 4.5 эта кнопка обведена красной линией.

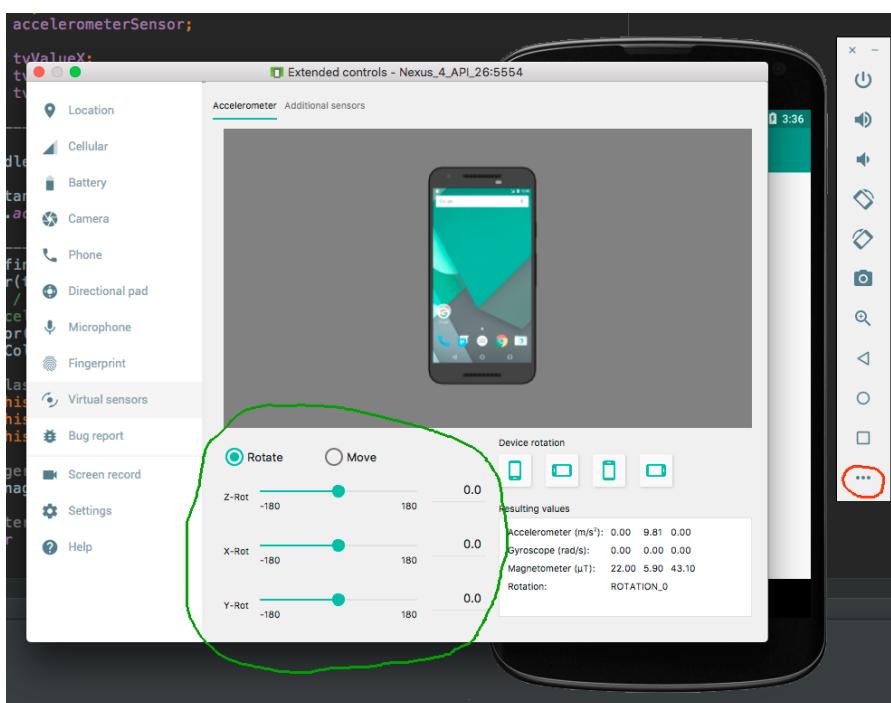


Рис. 4.5. Диалоговое окно «Extended Controls» с опциями изменения положения устройства (обведено зеленой линией)

Также на Рис. 4.5 зеленой линией обведены опции, при помощи которых можно менять положение эмули-

руемого устройства в пространстве с целью изменения получаемых от датчика акселерометра данных. На Рис. 4.6 в виде коллажа показано изменение положения эмулируемого устройства при помощи вышеописанных опций и данные от датчика акселерометра, которые получены в ответ на изменение положения устройства.

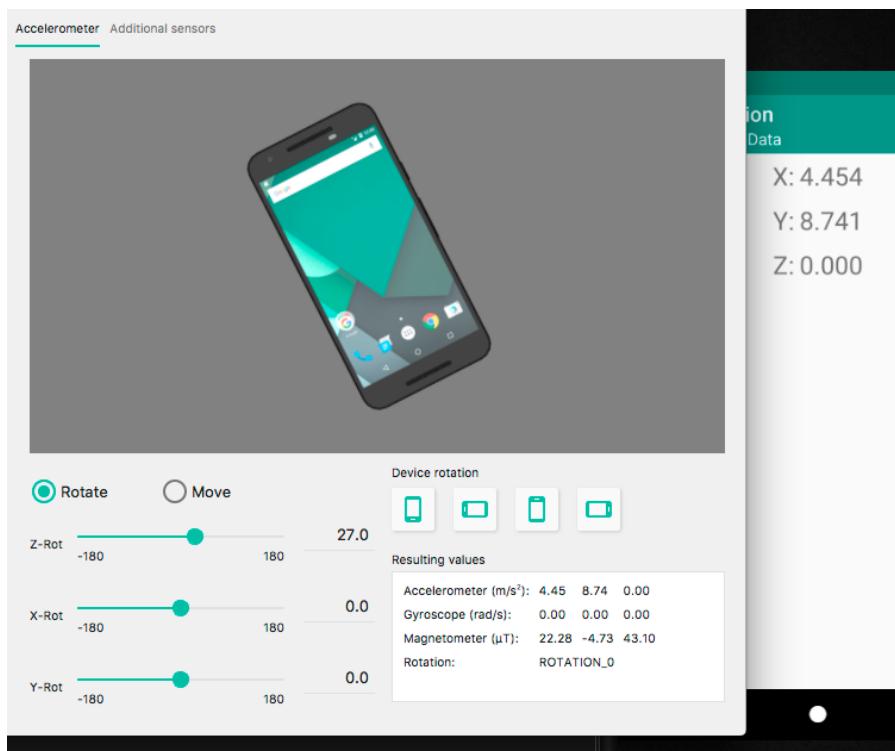


Рис. 4.6. Изменение получаемых от акселерометра данных в ответ на изменение положения устройства

Исходный код приложения из Листинга 4.5 находится в модуле «app10» проекта Android Studio, который прилагается к данному уроку.

5. Домашнее задание

Разработайте приложение, которое будет воспроизводить некоторый звуковой файл при помощи объекта `android.media.SoundPool`. Звуковой файл может быть любой — на ваше усмотрение. В этом приложении должна быть возможность настройки уровня громкости для этого воспроизводимого файла. Уровень громкости необходимо сохранять в файле Данных Предпочтений. Для задания величины уровня громкости необходимо разработать специальный (кастомный) виджет «Уровень громкости».

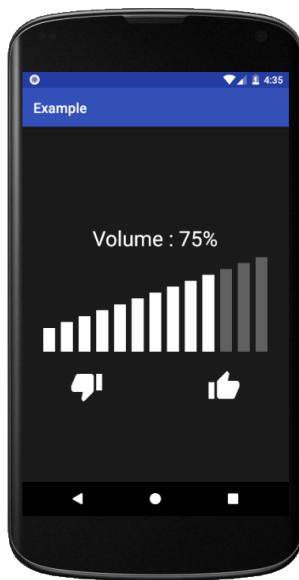


Рис. 5.1. Приблизительный внешний вид виджета «Уровень громкости»

Приблизительный внешний вид виджета «Уровень громкости» изображен на Рис. 5.1. Как видно из Рис. 5.1, уровень громкости должен регулироваться при помощи нажатия на иконки «Вниз» (рука с указательным пальцем вниз) и «Вверх» (рука с указательным пальцем вверх). Не обязательно иконки «Вниз» и «Вверх» должны выглядеть так, как показано на Рис. 5.1. У вас есть возможность проявить творческий подход при создании виджета «Уровень громкости». И этот творческий подход приветствуется.

Желаем вам удачи.

P.S. Не забывайте об экзаменационном задании, которое было выдано вам в прошлых уроках.



Урок № 8.

Воспроизведение аудиофайлов в Android приложениях. Класс android.media.SoundPool

© Бояршинов Юрий

© Компьютерная Академия «Шаг»

www.itstep.org

Все права на охраняемые авторским правом фото-, аудио- и видеопроизведения, фрагменты которых использованы в материале, принадлежат их законным владельцам. Фрагменты произведений используются в иллюстративных целях в объеме, оправданном поставленной задачей, в рамках учебного процесса и в учебных целях, в соответствии со ст. 1274 ч. 4 ГК РФ и ст. 21 и 23 Закона Украины «Про авторське право і суміжні права». Объем и способ цитируемых произведений соответствует принятым нормам, не наносит ущерба нормальному использованию объектов авторского права и не ущемляет законные интересы автора и правообладателей. Цитируемые фрагменты произведений на момент использования не могут быть заменены альтернативными, не охраняемыми авторским правом аналогами, и как таковые соответствуют критериям добросовестного использования и честного использования.

Все права защищены. Полное или частичное копирование материалов запрещено. Согласование использования произведений или их фрагментов производится с авторами и правообладателями. Согласованное использование материалов возможно только при указании источника.

Ответственность за несанкционированное копирование и коммерческое использование материалов определяется действующим законодательством Украины.