

```
<item name="smileyExpression">happy</item>
</style>
```

Он применяется в нашем `SmileyView` путем добавления его в качестве последнего параметра вызова функции `getStyledAttributes` (см. код в шаге 3):

```
TypedArray a = context.obtainStyledAttributes(attrs, R.styleable.SmileyView,
defStyleAttr, R.style.DefaultSmileyViewStyle);
```

Обратите внимание, что любые значения атрибутов, заданные в «раздутом» файле макета (см. код в шаге 2), будут переопределять соответствующие значения стиля по умолчанию.

5. (Необязательно) Предоставление стилей внутри тем: это делается путем добавления нового атрибута ссылки на стиль, который может быть использован внутри тем, и предоставления стиля для этого атрибута. В данном случае мы просто назвали наш атрибут ссылки `smileyStyle`:

```
<!-- attrs.xml -->
<attr name="smileyStyle" format="reference" />
```

Затем мы задаем стиль для темы нашего приложения (здесь мы просто используем стиль по умолчанию из шага 4):

```
<!-- themes.xml -->
<style name="AppTheme" parent="AppBaseTheme">
    <item name="smileyStyle">@style/DefaultSmileyStyle</item>
</style>
```

28.3. Советы по повышению производительности CustomView

Не выделяйте новые объекты в `onDraw`:

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    Paint paint = new Paint(); // Не выделять здесь место
}
```

Вместо того чтобы отрисовывать `drawable`-объекты на холсте:

```
drawable.setBounds(boundsRect);
drawable.draw(canvas);
```

...для ускорения отрисовки используйте растровое изображение:

```
canvas.drawBitmap(bitmap, srcRect, boundsRect, paint);
```

Не следует перерисовывать все представление, чтобы обновить только небольшую его часть. Вместо этого перерисуйте конкретную часть представления:

```
invalidate(boundToBeRefreshed);
```

Если в представлении выполняется непрерывная анимация, например, циферблат часов, показывающийся каждую секунду, то, по крайней мере, остановите анимацию в `onStop()` активности и запустите ее снова в `onStart()` активности.

Не производите никаких вычислений внутри метода `onDraw` представления, вместо этого следует завершить отрисовку перед вызовом `invalidate()`. Использование этой техники позволяет избежать выпадения кадров в представлении.

Вращение

Основными операциями представления являются `translate`, `rotate` и т. д. Практически каждый разработчик сталкивался с проблемой при использовании растровых изображений или градиенты в пользовательском представлении. Если в представлении будет показан повернутый вид и растровое изображение должно быть повернуто в этом пользовательском представлении, многие из нас подумают, что это будет ресурсоемко. Распространено мнение, что вращение растрового изображения очень затратно, поскольку для этого необходимо переворачивать пиксельную матрицу растрового изображения. Но на самом деле это не так сложно. Вместо того чтобы вращать растровое изображение, достаточно повернуть сам холст!

```
// Сохранить состояние холста
int save = canvas.save();
// Поверните холст, указав угол и центральную точку в качестве точки вращения
canvas.rotate(pivotX, pivotY, angle);
// Отрисовывайте что хотите
// В принципе, все, что здесь отрисовывается, будет отрисовано в соответствии с углом,
// на который вы повернули холст.
canvas.drawBitmap(...);
// Теперь верните холст в исходное состояние
canvas.restore(save);
// Пока холст не будет возвращен в исходное состояние, дальнейшая отрисовка
// также будет повернутой.
```

28.4. Создание составного представления

Составное представление (`compound view`) – это пользовательская группа `ViewGroup`, которая рассматривается окружающим программным кодом как одно представление. Такая `ViewGroup` может быть действительно полезна в так называемом DDD-подобном дизайне (Domain-driven design, см. https://en.wikipedia.org/wiki/Domain-driven_design), поскольку она может соответствовать агрегату, в данном примере – контакту (`Contact`). Она может быть повторно использована везде, где отображается контакт.

Это означает, что окружающий код контроллера – `Activity`, `Fragment` или `Adapter` – может просто передать объект данных в представление, не разделяя его на множество различных виджетов пользовательского интерфейса.

Это облегчает повторное использование кода и делает проектирование более качественным в соответствии с принципами объектно-ориентированных проектирования и программирования SOLID (см. [https://en.wikipedia.org/wiki/SOLID_\(object-oriented_design\)](https://en.wikipedia.org/wiki/SOLID_(object-oriented_design))).

Макет XML

Обычно с него и начинают. У вас фрагмент XML, который вы решили использовать повторно, возможно, в качестве `<include>`. Извлеките его в отдельный XML-файл и оберните корневой тег в элемент `<merge>`:

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

    <ImageView
        android:id="@+id/photo"
        android:layout_width="48dp"
        android:layout_height="48dp"
        android:layout_alignParentRight="true" />

    <TextView
        android:id="@+id/name"
        android:layout_width="match_parent"
```

```

        android:layout_height="wrap_content"
        android:layout_toLeftOf="@+id/photo" />

    <TextView
        android:id="@+id/phone_number"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/name"
        android:layout_toLeftOf="@+id/photo" />
</merge>
```

Этот XML-файл прекрасно работает в редакторе макетов в Android Studio. Вы можете обращаться с ним, как с любым другим макетом.

Составная группа ViewGroup

Получив XML-файл, создайте пользовательскую группу представлений:

```

import android.annotation.TargetApi;
import android.content.Context;
import android.os.Build;
import android.util.AttributeSet;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.RelativeLayout;
import android.widget.ImageView;
import android.widget.TextView;

import myapp.R;

/**
 * Составное представление для отображения контактов.
 */
/**
 * Этот класс может быть помещен в XML-макет или инстанцирован программно,
 * он будет работать корректно в любом случае.
 */
public class ContactView extends RelativeLayout {

    // Этот класс расширяет RelativeLayout, поскольку он поставляется с автоматическим
    // (MATCH_PARENT, MATCH_PARENT) макетом для своего дочернего элемента. Можно
    // расширить android.view.ViewGroup класс, если хотите получить больше контроля.

    // 1. Реализуйте конструкторы суперклассов.
    public ContactView(Context context) {
        super(context);
        init(context, null);
    }

    // два лишних конструктора оставлены в стороне для сокращения примера

    @TargetApi(Build.VERSION_CODES.LOLLIPOP)
    public ContactView(Context context, AttributeSet attrs, int defStyleAttr, int
        defStyleAttrRes) {
        super(context, attrs, defStyleAttr, defStyleAttrRes);
        init(context, attrs);
    }

    // 2. Инициализируем представление, "раздувая" XML, используя 'this' в качестве
    // родителя
```

```

private TextView mName;
private TextView mPhoneNumber;
private ImageView mPhoto;

private void init(Context context, AttributeSet attrs) {
    LayoutInflator.from(context).inflate(R.layout.contact_view, this, true);
    mName = (TextView) findViewById(R.id.name);
    mPhoneNumber = (TextView) findViewById(R.id.phone_number);
    mPhoto = (ImageView) findViewById(R.id.photo);
}

// 3. Определить сеттер, который выражается в вашей доменной модели.
// Весь код контроллера может просто вызывать этот сеттер вместо того, чтобы
// возиться с множеством строк, настроек видимости, цветов, анимации и т. д.
// Если вы не используете пользовательского представления, этот код обычно
// оказывается в статическом вспомогательном методе (что плохо) или копии
// этого кода будут скопированы-вставлены повсюду (еще хуже)

public void setContact(Contact contact) {
    mName.setText(contact.getName());
    mPhoneNumber.setText(contact.getPhoneNumber());
    if (contact.hasPhoto()) {
        mPhoto.setVisibility(View.VISIBLE);
        mPhoto.setImageBitmap(contact.getPhoto());
    } else {
        mPhoto.setVisibility(View.GONE);
    }
}
}

```

Метод `init(Context, AttributeSet)` – это метод, в которомчитываются все пользовательские XML-атрибуты, как было описано в главе «Добавление атрибутов в представления».

После того как все эти элементы установлены, их можно использовать в своем приложении.

Использование в XML

Приведем пример `fragment_contact_info.xml`, иллюстрирующий размещение одного `ContactView` поверх списка сообщений:

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <!-- Составное представление становится похожим на любой другой XML-элемент
    представления -->
    <myapp.ContactView
        android:id="@+id/contact"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

    <android.support.v7.widget.RecyclerView
        android:id="@+id/message_list"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"/>

</LinearLayout>

```

Использование в коде

Приведем пример RecyclerView.Adapter, который отображает список контактов. Этот пример показывает, насколько чище становится код контроллера, когда в нем полностью отсутствуют манипуляции с представлениями:

```
package myapp;

import android.content.Context;
import android.support.v7.widget.RecyclerView;
import android.view.ViewGroup;

public class ContactsAdapter extends RecyclerView.Adapter<ContactsViewHolder> {

    private final Context context;

    public ContactsAdapter(final Context context) {
        this.context = context;
    }

    @Override
    public ContactsViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        ContactView v = new ContactView(context); // <--- this
        return new ContactsViewHolder(v);
    }

    @Override
    public void onBindViewHolder(ContactsViewHolder holder, int position) {
        Contact contact = this.getItem(position);
        holder.setContact(contact); // <--- this
    }

    static class ContactsViewHolder extends RecyclerView.ViewHolder {

        public ContactsViewHolder(ContactView itemView) {
            super(itemView);
        }

        public void setContact(Contact contact) {
            ((ContactView) itemView).setContact(contact); // <--- this
        }
    }
}
```

28.6. Реакция на события касания

Многие пользовательские представления должны воспринимать взаимодействие с пользователем в виде событий касания. Получить доступ к событиям касания можно, переопределив onTouchEvent. Существует ряд действий, которые можно отфильтровать. Основными из них являются:

- ACTION_DOWN: срабатывает один раз, когда палец впервые касается экрана.
- ACTION_MOVE: Вызывается каждый раз, когда ваш палец немного перемещается по экрану. Она вызывается много раз.
- ACTION_UP: Это последнее действие, которое будет вызвано при отрыве пальца от экрана.

Вы можете добавить в представление следующий метод и затем наблюдать за выводом журнала при касании и перемещении пальца по представлению.

```

@Override
public boolean onTouchEvent(MotionEvent event) {
    int x = (int) event.getX();
    int y = (int) event.getY();
    int action = event.getAction();

    switch (action) {
        case MotionEvent.ACTION_DOWN:
            Log.i("CustomView", "onTouchEvent: ACTION_DOWN: x = " + x + ", y = " + y);
            break;

        case MotionEvent.ACTION_MOVE:
            Log.i("CustomView", "onTouchEvent: ACTION_MOVE: x = " + x + ", y = " + y);
            break;

        case MotionEvent.ACTION_UP:
            Log.i("CustomView", "onTouchEvent: ACTION_UP: x = " + x + ", y = " + y);
            break;
    }
    return true;
}

```

Дополнительную информацию можно получить в официальной документации по Android в разделе «Реакция на события касания»: <https://developer.android.com/training/graphics/opengl/touch.html>.

Глава 29. Получение расчетных размеров представления

29.1. Расчет начальных размеров вида в активности

```

package com.example;

import android.os.Bundle;
import android.support.annotation.Nullable;
import android.util.Log;
import android.view.View;
import android.view.ViewTreeObserver;

public class ExampleActivity extends Activity {

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

        final View viewToMeasure = findViewById(R.id.view_to_measure);

        // размеры viewToMeasure на данный момент неизвестны.
        // viewToMeasure.getWidth() и viewToMeasure.getHeight() оба возвращают 0,
        // независимо от размеров на экране.
    }
}

```

```
viewToMeasure.getViewTreeObserver().addOnPreDrawListener(new ViewTreeObserver.OnPreDrawListener() {
    @Override
    public boolean onPreDraw() {
        // viewToMeasure теперь измерен и размещен, а отображаемые размеры известны.
        logComputedViewDimensions(viewToMeasure.getWidth(), viewToMeasure.getHeight());
        // Удалите этот слушатель, поскольку теперь мы успешно вычислили
        // нужные размеры.
        viewToMeasure.getViewTreeObserver().removeOnPreDrawListener(this);

        // Для продолжения отрисовки всегда возвращается true.
        return true;
    }
});

private void logComputedViewDimensions(final int width, final int height) {
    Log.d("example", "viewToMeasure имеет ширину " + width);
    Log.d("example", "viewToMeasure имеет высоту " + height);
}
```

Глава 30. Добавление FuseView в проект Android

Экспортируйте Fuse.View с сайта fusetools (<https://www.fusetools.com/>) и используйте его в существующем проекте Android. Наша цель – экспортировать пример приложения hikr (<https://github.com/fusetools/hikr>) и использовать его внутри активности. Готовую работу можно найти по адресу <https://github.com/lucamtudor/hikr-fuse-view>.

30.1. Приложение hikr, просто еще один android.view.View

Предварительные условия

- У вас должен быть установлен fuse (<https://www.fusetools.com/downloads>)
- Вы должны выполнить вводный урок (<https://www.fusetools.com/docs/tutorial/tutorial>)
- В терминале следует выполнить: `fuse install android`
- Затем: `uno install Fuse.Views`

Шаг 1:

```
git clone https://github.com/fusetools/hikr
```

Шаг 2: Добавить ссылку на пакет в Fuse.Views

Найдите в корневом каталоге проекта файл `hikr.unoproj` и добавьте в массив "Packages" файл "Fuse.Views".

```
"RootNamespace": "",
```

```

    "Packages": [
        "Fuse",
        "FuseJS",
        "Fuse.Views"
    ],
    "Includes": [
        "*",
        "Modules/*.js:Bundle"
    ]
}

```

Шаг 3: Создание компонента HikrApp для хранения всего приложения

В корневой папке проекта создайте новый файл с именем `HikrApp.ux` и вставьте в него содержимое файла `MainView.ux`.

`HikrApp.ux:`

```

<App Background="#022328">
    <iOS.StatusBarConfig Style="Light" />
    <Android.StatusBarConfig Color="#022328" />

    <Router ux:Name="router" />

    <ClientPanel>
        .
        <Navigator DefaultPath="splash">
            <SplashPage ux:Template="splash" router="router" />
            <HomePage ux:Template="home" router="router" />
            <EditHikePage ux:Template="editHike" router="router" />
        </Navigator>
    </ClientPanel>
</App>

```

В `HikrApp.ux`:

- замените `<App>` теги на `<Page>`
- добавьте `ux:Class="HikrApp"` в открывающуюся `<Page>`
- удалите `<ClientPanel>`, и нам больше не придется беспокоиться о строке состояния или кнопках навигации внизу

`HikrApp.ux:`

```

<Page ux:Class="HikrApp" Background="#022328">
    <iOS.StatusBarConfig Style="Light" />
    <Android.StatusBarConfig Color="#022328" />

    <Router ux:Name="router" />

    <Navigator DefaultPath="splash">
        <SplashPage ux:Template="splash" router="router" />
        <HomePage ux:Template="home" router="router" />
        <EditHikePage ux:Template="editHike" router="router" />
    </Navigator>
</Page>

```

Используйте только что созданный компонент `HikrApp` в файле `MainView.ux`.

Замените содержимое файла `MainView.ux` на:

```
<App>
    <HikrApp/>
</App>
```

Наше приложение вернулось к своему обычному поведению, но теперь мы вынесли его в отдельный компонент под названием HikrApp.

Шаг 4: Внутри MainView.ux замените теги **<App>** на **<ExportedViews>** и добавьте **ux:Template="HikrAppView"** к **<HikrApp />**

```
<ExportedViews>
    <HikrApp ux:Template="HikrAppView" />
</ExportedViews>
```

Запомните шаблон HikrAppView, поскольку он понадобится нам для получения ссылки на наше представление из Java.

Примечание: из документации по Fuse:

Экспортированные Views будут вести себя как App при обычном предварительном просмотре fuse preview и сборке uno.

Это не так. Вы получите эту ошибку при предварительном просмотре из Fuse Studio:

Ошибка: Не удалось найти тег App ни в одном из включенных UX-файлов. Не забыли ли вы включить UX-файл, содержащий тег App?

Шаг 5: Оберните **<DockPanel>** файла SplashPage.ux в **<GraphicsView>**

```
<Page ux:Class="SplashPage">
    <Router ux:Dependency="router" />

    <JavaScript File="SplashPage.js" />

    <GraphicsView>
        <DockPanel ClipToBounds="true">
            <Video Layer="Background" File="../Assets/nature.mp4" IsLooping="true"
                AutoPlay="true" StretchMode="UniformToFill" Opacity="0.5">
                <Blur Radius="4.75" />
            </Video>

            <hikr.Text Dock="Bottom" Margin="10" Opacity=".5" TextAlignment="Center"
                FontSize="12">original video by Graham Uhelski</hikr.Text>

            <Grid RowCount="2">
                <StackPanel Alignment="VerticalCenter">
                    <hikr.Text Alignment="HorizontalCenter" FontSize="70">hikr</hikr.
                    Text>
                    <hikr.Text Alignment="HorizontalCenter" Opacity=".5">get out
                    there</hikr.Text>
                </StackPanel>

                <hikr.Button Text="Get Started" FontSize="18" Margin="50,0"
                    Alignment="VerticalCenter" Clicked="{goToHomePage}" />
            </Grid>
        </DockPanel>
    </GraphicsView>
</Page>
```

Шаг 6: Экспорт проекта fuse в виде библиотеки aar

- в терминале, в папке корневого проекта выполните: uno clean

- в терминале, в корневой папке проекта выполните: uno build -t=android -DLIBRARY

Шаг 7: Подготовка проекта Android:

- скопируйте aar из .../rootHikeProject/build/Android/Debug/app/build/outputs/aar/app-debug.aar в .../androidRootProject/app/libs
- добавьте flatDir { dirs 'libs' } в корневой файл build.gradle

// Файл сборки верхнего уровня, в который можно добавить параметры конфигурации,
// общие для всех подпроектов/модулей.

```
buildscript { ... }
```

```
...
allprojects {
    repositories {
        jcenter()
        flatDir {
            dirs 'libs'
        }
    }
}
```

- добавить compile(name: 'app-debug', ext: 'aar') к зависимостям в app/build.gradle

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

```
android {
    compileSdkVersion 25
    buildToolsVersion "25.0.2"
    defaultConfig {
        applicationId "com.shiftstudio.fuseviewtest"
        minSdkVersion 16
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
rules.pro'
        }
    }
}
```

```
dependencies {
    compile(name: 'app-debug', ext: 'aar')
    compile fileTree(dir: 'libs', include: ['*.jar']) androidTestCompile('com.
    android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
}
```

```
compile 'com.android.support:appcompat-v7:25.3.1'
testCompile 'junit:junit:4.12'
}
```

- добавьте следующие свойства к активности внутри AndroidManifest.xml:

```
android:launchMode="singleTask"
android:taskAffinity=""
android:configChanges="orientation|keyboardHidden|screenSize|smallestScreenSize"
```

Ваш AndroidManifest.xml будет выглядеть следующим образом:

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:launchMode="singleTask"
            android:taskAffinity=""
            android:configChanges="orientation|keyboardHidden|screenSize|
smallestScreenSize">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Шаг 8: Отображение Fuse.View HikrAppView в Activity:

- обратите внимание, что ваша Activity должна наследовать FuseViewsActivity

```
public class MainActivity extends FuseViewsActivity {

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

        final ViewHandle fuseHandle = ExportedViews.instantiate("HikrAppView");

        final FrameLayout root = (FrameLayout) findViewById(R.id.fuse_root);
        final View fuseApp = fuseHandle.getView();
        root.addView(fuseApp);
    }
}
```

activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.shiftstudio.fuseviewtest.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_gravity="center_horizontal"
        android:textSize="24sp"
        android:textStyle="bold"
        android:layout_height="wrap_content"
        android:text="Hello World, from Kotlin" />

    <FrameLayout
        android:id="@+id/fuse_root"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:layout_width="wrap_content"
            android:text="THIS IS FROM NATIVE.\nBEHIND FUSE VIEW"
            android:layout_gravity="center"
            android:textStyle="bold"
            android:textSize="30sp"
            android:background="@color/colorAccent"
            android:textAlignment="center"
            android:layout_height="wrap_content" />

    </FrameLayout>
</LinearLayout>
```

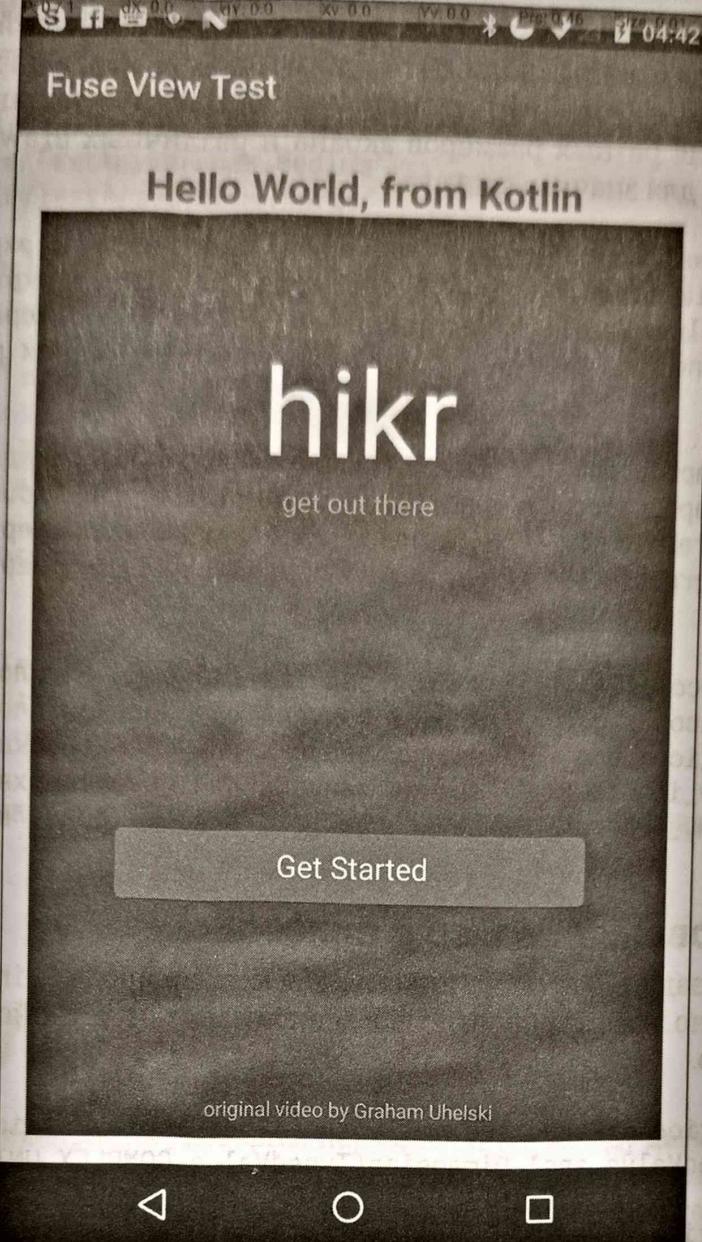
Примечание: при нажатии кнопки “назад” на Android происходит сбой приложения. Проблема обсуждалась на форуме fuse (https://www.fusetools.com/community/forums/bug-reports/fuseview_problems).

A/libc: Fatal signal 11 (SIGSEGV), code 1, fault addr 0xdeadcab1 in tid 18026
(io.fuseviewtest)

[05-25 11:52:33.658 16567:16567 W/]

debuggerd: handling request: pid=18026 uid=10236 gid=10236 tid=18026

И в итоге получается примерно такой результат. Небольшой ролик можно также найти на github: <https://github.com/lucamtudor/hikr-fuse-view/blob/master/fuse-fun.mp4>.



Глава 31. Поддержка экранов с различными разрешениями и размерами

31.1. Использование квалификаторов конфигурации

Android поддерживает несколько квалификаторов конфигурации, позволяющих управлять выбором альтернативных ресурсов в зависимости от характеристик текущего экрана устройства. Конфигурационный квалификатор – это строка, которую можно добавить к каталогу ресурсов в проекте Android и которая определяет конфигурацию, для которой предназначены ресурсы, находящиеся в нем.

Для использования конфигурационного квалификатора:

1. Создайте новый каталог в каталоге res/ вашего проекта и назовите его в формате: <имя_ресурса>-<квалификатор>. <имя_ресурса> – стандартное имя ресурса (например, drawable или layout).

2. **<квалификатор>** – это квалификатор конфигурации, указывающий конфигурацию экрана, для которого должны использоваться эти ресурсы (например, hdpi или xlarge).

Например, следующие каталоги ресурсов приложений предоставляют различные варианты компоновки для разных размеров экрана и различных drawable-объектов. Папки **mipmap/** используются для значков запуска.

```
res/layout/my_layout.xml           // макет для нормального размера экрана ("по умолчанию")
res/layout-large/my_layout.xml    // макет для большого размера экрана
res/layout-xlarge/my_layout.xml  // макет для сверхбольшого размера экрана
res/layout-xlarge-land/my_layout.xml // макет для сверхбольших размеров в альбомной
                                    // ориентации

res/drawable-mdpi/graphic.png   // раcтровое изображение для средней плотности
res/drawable-hdpi/graphic.png   // раcтровое изображение для высокой плотности
res/drawable-xhdpi/graphic.png  // раcтровое изображение для сверхвысокой плотности
res/drawable-xxhdpi/graphic.png // раcтровое изображение для сверхсверхвысокой
                                    // плотности

res/mipmap-mdpi/my_icon.png     // значок запуска для средней плотности
res/mipmap-hdpi/my_icon.png     // значок запуска для высокой плотности
res/mipmap-xhdpi/my_icon.png   // значок запуска для сверхвысокой плотности
res/mipmap-xxhdpi/my_icon.png  // значок запуска для сверхсверхвысокой плотности
res/mipmap-xxxhdpi/my_icon.png // значок запуска для еще более высокой плотности
```

31.2. Преобразование dp и sp в пиксели

Если необходимо задать значение пикселя для чего-то вроде `Paint.setTextSize`, но при этом нужно, чтобы оно масштабировалось в зависимости от устройства, можно преобразовать значения `dp` и `sp`.

```
DisplayMetrics metrics = Resources.getSystem().getDisplayMetrics();
float pixels = TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_SP, 12f, metrics);

DisplayMetrics metrics = Resources.getSystem().getDisplayMetrics();
float pixels = TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP, 12f, metrics);
```

В качестве альтернативы можно преобразовать ресурс размерности в пиксели, если у вас есть контекст, из которого загружается ресурс:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="size_in_sp">12sp</dimen>
    <dimen name="size_in_dp">12dp</dimen>
</resources>

// Получение точного размера, указанного ресурсом
float pixels = context.getResources().getDimension(R.dimen.size_in_sp);
float pixels = context.getResources().getDimension(R.dimen.size_in_dp);

// Получение размерности, указанной ресурсом, для использования в качестве размера.
// Значение округляется до ближайшего целого числа, но не менее 1px
int pixels = context.getResources().getDimensionPixelSize(R.dimen.size_in_sp);
int pixels = context.getResources().getDimensionPixelSize(R.dimen.size_in_dp);

// Получение размерности, указанной ресурсом, для использования в качестве смещения
// Значение округляется до ближайшего целого числа и может быть равно 0px
int pixels = context.getResources().getDimensionPixelOffset(R.dimen.size_in_sp);
int pixels = context.getResources().getDimensionPixelOffset(R.dimen.size_in_dp);
```

31.3. Размер текста и различные размеры экрана Android

Иногда лучше иметь только три варианта:

```
style="@android:style/TextAppearance.Small"
style="@android:style/TextAppearance.Medium"
style="@android:style/TextAppearance.Large"
```

Используйте small и large для отличия от обычного размера экрана:

```
<TextView
    android:id="@+id/TextViewTopBarTitle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@android:style/TextAppearance.Small"/>
```

Для обычного режима ничего указывать не нужно:

```
<TextView
    android:id="@+id/TextViewTopBarTitle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

Это позволяет избежать тестирования и уточнения размеров для разных размеров экрана.

Глава 32. ViewFlipper

ViewFlipper – это ViewAnimator, который переключается между двумя или более видами, добавленными к нему. Одновременно отображается только один дочерний вид. По желанию пользователя ViewFlipper может автоматически переходить от одного дочернего элемента к другому с регулярным интервалом.

32.1. ViewFlipper со скольжением изображения

XML-файл:

```
<ViewFlipper
    android:id="@+id/viewflip"
    android:layout_width="match_parent"
    android:layout_height="250dp"
    android:layout_weight="1"/>
/>
```

Java-код:

```
public class BlankFragment extends Fragment{
    ViewFlipper viewFlipper;
    FragmentManager fragmentManager;
    int gallery_grid_Images[] = {drawable.image1, drawable.image2, drawable.image3,
        drawable.image1, drawable.image2, drawable.image3, drawable.image1, drawable.
        image2, drawable.image3, drawable.image1
    };

    @Override
```

```

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    View rootView = inflater.inflate(fragment_blank, container, false);
    viewFlipper = (ViewFlipper)rootView.findViewById(R.id.viewflip);
    for(int i=0; i<gallery_grid_Images.length; i++){
        // Это позволит создать динамические представления изображений и добавить
        // их в ViewFlipper.
        setFlipperImage(gallery_grid_Images[i]);
    }
    return rootView;
}

private void setFlipperImage(int res) {
    Log.i("Set Filpper Called", rest");
    ImageView image = new ImageView(getContext());
    image.setBackgroundResource(res);
    viewFlipper.addView(image);
    viewFlipper.setFlipInterval(1000);
    viewFlipper.setAutoStart(true);
}
}

```

Глава 33. Паттерны проектирования

Паттерны проектирования – это формализованные лучшие практики, которые программист может использовать для решения общих проблем при проектировании приложения или системы. Паттерны проектирования позволяют ускорить процесс разработки, предоставляя проверенные парадигмы разработки. Повторное использование этих паттернов помогает предотвратить мелкие недочеты, которые могут привести к серьезным проблемам, а также улучшает читаемость кода для разработчиков, знакомых с паттернами.

33.1. Шаблон наблюдателя

Шаблон наблюдателя (observer pattern) – это распространенный шаблон, который широко используется во многих контекстах. В качестве примера можно привести YouTube: если вам нравится какой-то канал и вы хотите получать все новости и смотреть новые видеоролики с этого канала, вам необходимо подписаться на этот канал. Тогда при публикации новостей на этом канале вы (и все остальные подписчики) будете получать уведомление.

Наблюдатель состоит из двух компонентов. Один из них – это вещатель (канал), а другой – приемник (вы или любой другой подписчик). Вещатель будет обрабатывать все экземпляры приемников, которые подписались на него. Когда вещатель запускает новое событие, он сообщает об этом всем экземплярам приемника. Когда приемник получает событие, он должен отреагировать на него, например, включить YouTube и воспроизвести новое видео.

Реализация шаблона наблюдателя

1. Вещатель должен предоставлять методы, позволяющие получателям подписываться и отписываться от него. Когда вещатель запускает событие, подписчики должны быть уведомлены о том, что событие произошло:

```
class Channel{
```

```

private List<Subscriber> subscribers;
public void subscribe(Subscriber sub) {
    // Добавление нового подписчика.
}
public void unsubscribe(Subscriber sub) {
    // Удалить подписчика.
}
public void newEvent() {
    // Событие уведомления для всех подписчиков.
}
}

```

2. Приемник должен реализовать метод, обрабатывающий событие от вещателя:

```

interface Subscriber {
    void doSubscribe(Channel channel);
    void doUnsubscribe(Channel channel);
    void handleEvent(); // Обработка нового события.
}

```

33.2. Пример класса Singleton

Паттерн Java Singleton

Для реализации паттерна Singleton используются различные подходы, но все они имеют следующие общие концепции.

- Частный конструктор, ограничивающий инстанцирование класса от других классов.
- Частная статическая переменная того же класса, которая является единственным экземпляром класса.
- Открытый статический метод, возвращающий экземпляр класса, является глобальной точкой доступа для внешнего мира для получения экземпляра класса singleton.

```

/**
 * Класс Singleton.
 */
public final class Singleton {
    /**
     * Частный конструктор, чтобы никто не мог инстанцировать класс.
     */
    private Singleton() {}

    /**
     * Статический к экземпляру класса.
     */
    private static final Singleton INSTANCE = new Singleton();

    /**
     * Вызывается пользователем для получения экземпляра класса.
     */
    @return экземпляр синглтона.
    */
    public static Singleton getInstance() {
        return INSTANCE;
    }
}

```

Глава 34. Активность

Параметр	Подробности
Intent	Может использоваться с функцией <code>startActivity</code> для запуска Activity
Bundle	Отображение ключей <code>String</code> на различные значения <code>Parcelable</code>
Context	Интерфейс для получения глобальной информации о среде приложения

Активность представляет собой отдельный экран с **пользовательским интерфейсом (UI)**. В приложении для Android может быть несколько активностей, например, в приложении для работы с электронной почтой может быть одна активность для просмотра списка всех писем, другая – для отображения содержимого письма, третья – для создания нового письма. Все активности в приложении работают вместе, создавая идеальный пользовательский опыт.

34.1. Режим запуска активности

Режим запуска определяет поведение новой или существующей активности в задаче. Существуют следующие возможные режимы запуска:

- standard
- singleTop
- singleTask
- singleInstance

Режим должен быть определен в манифесте android в элементе `<activity>` как атрибут `android:launchMode`.

```
<activity
    android:launchMode=["standard" | "singleTop" | "singleTask" | "singleInstance"] />
```

Стандарт:

Значение по умолчанию. Если этот режим установлен, то для каждой новой цели всегда будет создаваться новая активность. Таким образом, может получиться много активностей одного типа. Новая активность будет размещаться в верхней части задачи. Для разных версий Android есть некоторая разница: если активность запускается из другого приложения, то на Android ≤ 4.4 она будет размещена на той же задаче, что и стартовое приложение, а на Android ≥ 5.0 будет создана новая задача.

SingleTop:

Этот режим практически не отличается от стандартного. Может быть создано множество экземпляров активности `singleTop`. Отличие заключается в том, что если на вершине текущего стека уже существует экземпляр активности, то вместо создания нового экземпляра будет вызвана функция `onNewIntent()`.

SingleTask:

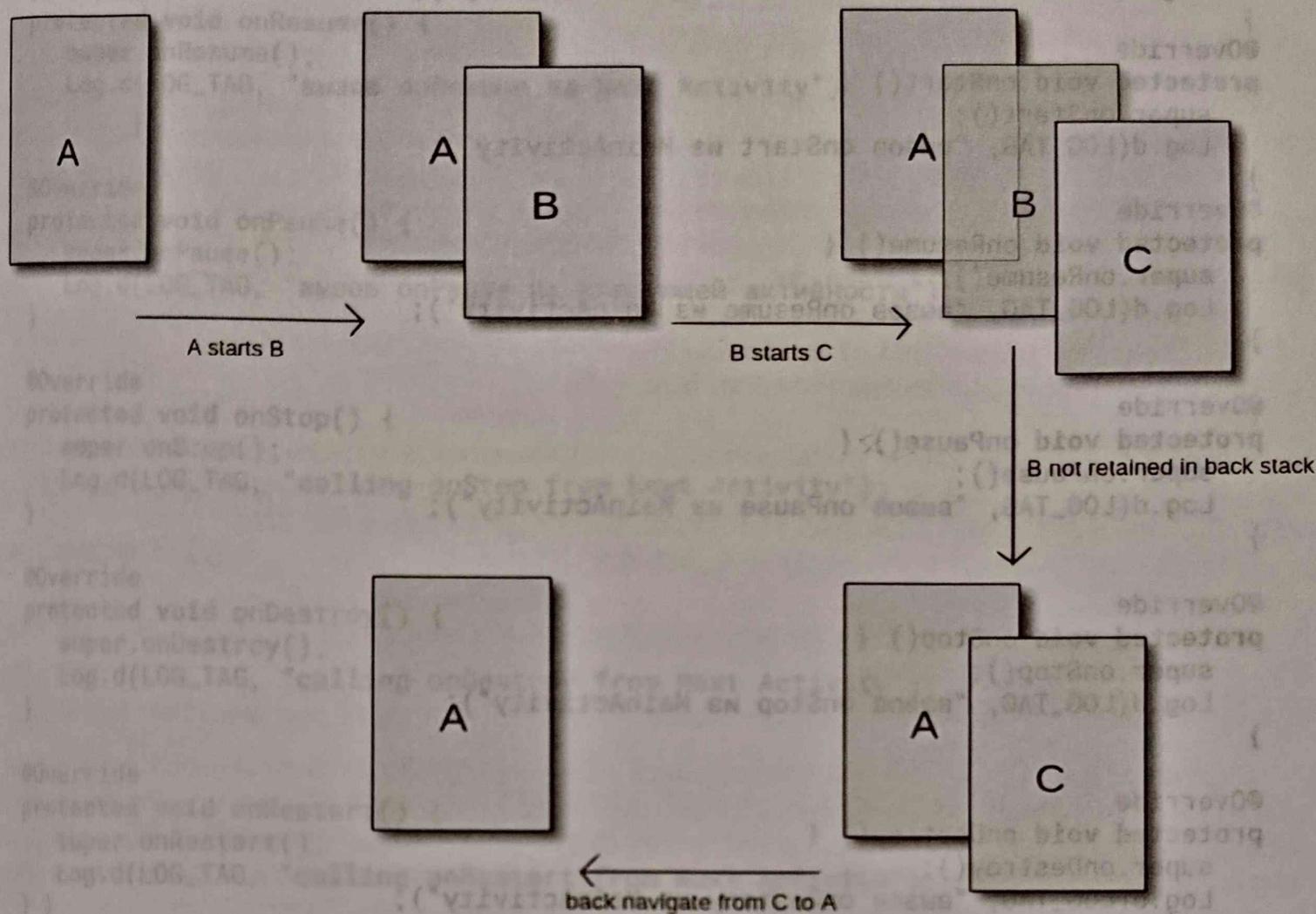
Активность с таким режимом запуска может иметь только один экземпляр в системе. В случае отсутствия новой задачи для активности будет создана новая задача. В противном случае задача с активностью будет перемещена на передний план и будет вызван `onNewIntent`.

SingleInstance:

Этот режим аналогичен режиму `singleTask`. Разница заключается в том, что задача, содержащая активность с `singleInstance`, может содержать только эту активность и ничего более. Когда активность `singleInstance` создаст другую активность, будет создана новая задача для размещения этой активности.

34.2. Исключение активности из истории обратного стека

Пусть есть активность B, которая может быть открыта и в дальнейшем может запускать другие активности. Но пользователь не должен сталкиваться с этим при переходе назад в активностях задания.



Наиболее простым решением является установка атрибута `noHistory` в `true` для данного тега `<activity>` в файле `AndroidManifest.xml`:

```
<activity
    android:name=".B"
    android:noHistory="true">
```

Такое же поведение возможно и из кода, если B вызовет `finish()` перед началом следующей активности:

```
finish();
startActivity(new Intent(context, C.class));
```

Обычно флаг `noHistory` используется для “всплывающего экрана” или действий при входе в систему.

34.3. Жизненный цикл активности в Android

Предположим, что в приложении есть `MainActivity`, которая может вызывать `NextActivity` по нажатию кнопки:

```
public class MainActivity extends AppCompatActivity {
```

```
private final String LOG_TAG = MainActivity.class.getSimpleName();
```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Log.d(LOG_TAG, "вызов onCreate из MainActivity");
}

@Override
protected void onStart() {
    super.onStart();
    Log.d(LOG_TAG, "вызов onStart из MainActivity");
}

@Override
protected void onResume() {
    super.onResume();
    Log.d(LOG_TAG, "вызов onResume из MainActivity");
}

@Override
protected void onPause() {
    super.onPause();
    Log.d(LOG_TAG, "вызов onPause из MainActivity");
}

@Override
protected void onStop() {
    super.onStop();
    Log.d(LOG_TAG, "вызов onStop из MainActivity");
}

@Override
protected void onDestroy() {
    super.onDestroy();
    Log.d(LOG_TAG, "вызов onDestroy из MainActivity");
}

@Override
protected void onRestart() {
    super.onRestart();
    Log.d(LOG_TAG, "вызов onRestart из MainActivity");
}

public void toNextActivity(){
    Log.d(LOG_TAG, "calling Next Activity");
    Intent intent = new Intent(this, NextActivity.class);
    startActivity(intent);
}

public class NextActivity extends AppCompatActivity {
    private final String LOG_TAG = NextActivity.class.getSimpleName();
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_next);
        Log.d(LOG_TAG, "вызов onCreate из Next Activity");
    }

    @Override
    protected void onStart() {
}

```

```

super.onStart();
Log.d(LOG_TAG, "вызов onStart из Next Activity");
}

@Override
protected void onResume() {
    super.onResume();
    Log.d(LOG_TAG, "вызов onResume из Next Activity");
}

@Override
protected void onPause() {
    super.onPause();
    Log.d(LOG_TAG, "вызов onPause из следующей активности");
}

@Override
protected void onStop() {
    super.onStop();
    Log.d(LOG_TAG, "calling onStop from Next Activity");
}

@Override
protected void onDestroy() {
    super.onDestroy();
    Log.d(LOG_TAG, "calling onDestroy from Next Activity");
}

@Override
protected void onRestart() {
    super.onRestart();
    Log.d(LOG_TAG, "calling onRestart from Next Activity");
}
}

```

При первом создании приложения:

вызываются:

D/MainActivity: вызов onCreate из MainActivity
D/MainActivity: вызов onStart из MainActivity
D/MainActivity: вызов onResume из MainActivity

Когда экран переходит в спящий режим:

вызываются:

08:11:03.142 D/MainActivity: вызов onPause из MainActivity
08:11:03.192 D/MainActivity: вызов onStop из MainActivity. И снова при пробуждении
08:11:55.922 D/MainActivity: вызов onRestart из MainActivity
08:11:55.962 D/MainActivity: вызов onStart из MainActivity
08:11:55.962 D/MainActivity: вызов onResume из MainActivity вызваны

Случай 1: Когда Next Activity вызывается из Main Activity

D/MainActivity: вызов Next Activity
D/MainActivity: вызов onPause из MainActivity
D/NextActivity: вызов onCreate из Next Activity
D/NextActivity: вызов onStart из Next Activity
D/NextActivity: вызов onResume из Next Activity
D/MainActivity: вызов onStop из MainActivity

При возврате в Main Activity из NextActivity с помощью кнопки назад

D/NextActivity: вызов onPause из NextActivity
 D/MainActivity: вызов onRestart из MainActivity
 D/MainActivity: вызов onStart из MainActivity
 D/MainActivity: вызов onResume из MainActivity
 D/NextActivity: вызов onStop из NextActivity
 D/NextActivity: вызов onDestroy из NextActivity

Случай 2: Когда активность частично скрыта (при нажатии кнопки обзора) или когда приложение переходит в фоновый режим, а другое приложение полностью закрывает его:

D/MainActivity: вызов onPause из MainActivity
 D/MainActivity: вызов onStop из MainActivity
 и когда приложение снова окажется на переднем плане, готовое принимать входные данные пользователя, вызываются:
 D/MainActivity: вызов onRestart из MainActivity
 D/MainActivity: вызов onStart из MainActivity
 D/MainActivity: вызов onResume из MainActivity

Случай 3: Когда действие вызывается для выполнения неявного намерения и пользователь должен сделать выбор. Например, при нажатии кнопки “Поделиться” пользователю необходимо выбрать приложение из отображаемого списка приложений.

D/MainActivity: вызов onPause из MainActivity

Активность видна, но сейчас не активна. Когда выбор будет завершен и приложение станет активным, вызывается:

D/MainActivity: вызов onResume из MainActivity

Случай 4: Когда приложение завершается в фоновом режиме (чтобы освободить ресурсы для другого приложения переднего плана), последним вызовом перед завершением приложения будет или onStop. onCreate и onDestroy будут вызываться максимум один раз при каждом запуске приложения. А вот onPause, onStop, onRestart, onStart, onResume могут вызываться много раз в течение жизненного цикла приложения.

34.4. Завершение работы приложения с исключением из последних открытых

Сначала определите ExitActivity в файле AndroidManifest.xml:

```
<activity
    android:name="com.your_example_app.activities.ExitActivity"
    android:autoRemoveFromRecents="true"
    android:theme="@android:style/Theme.NoDisplay" />
```

Класс ExitActivity:

```
/**
 * Активность для выхода из приложения, не оставаясь в стеке последних открытых
 * приложений
 */
public class ExitActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        if (Utils.hasLollipop()) {
            finishAndRemoveTask();
        } else if (Utils.hasJellyBean()) {
```

```

        finishAffinity();
    } else {
        finish();
    }
}
/***
 * Выход из приложения и исключение из Recents (списка последних)
 *
 * @param context Контекст для использования
 */
public static void exitApplication(ApplicationContext context) {
    Intent intent = new Intent(context, ExitActivity.class);
    intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_
    TASK | Intent.FLAG_ACTIVITY_NO_ANIMATION | Intent.FLAG_ACTIVITY_EXCLUDE_FROM_
    RECENTS);
    context.startActivity(intent);
}
}

```

34.5. Представление пользовательского интерфейса с помощью setContentView

Класс Activity позаботится о создании для вас окна, в котором вы можете разместить свой пользовательский интерфейс с помощью setContentView. Существует три метода setContentView:

- `setContentView(int layoutResID)` – установка содержимого активности из ресурса макета
- `setContentView(View view)` – установка содержимого активности в явное представление
- `setContentView(View view, ViewGroup.LayoutParams params)` – установка содержимого активности в явное представление с указанными параметрами.

При вызове setContentView это представление помещается непосредственно в иерархию представлений активности. Оно само может быть сложной иерархией представлений.

ПРИМЕРЫ

Установите содержимое из файла ресурсов:

Добавьте файл ресурсов (в данном примере main.xml) с иерархией представлений:

```

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

    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello" />

</FrameLayout>

```

Установите его в качестве содержимого в активности:

```

public final class MainActivity extends Activity {

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

```

```

    // Ресурс будет раздуваться,
    // добавляя все представления верхнего уровня в активность.
    setContentView(R.layout.main);
}
}

```

Установите содержимое в явное представление:

```
public final class MainActivity extends Activity {
```

```

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

        // Создание представления с контейнером
        final FrameLayout root = new FrameLayout(this);
        final TextView text = new TextView(this);
        text.setText("Hello");
        root.addView(text);

        // Установить контейнер в качестве представления содержимого
        setContentView(root);
    }
}
```

34.6. Навигация вверх для активностей

Навигация вверх осуществляется в Android путем добавления в Manifest.xml к тегу activity тега android:parentActivityName=". ". В основном с помощью этого тега вы сообщаете системе о родительской активности данной активности.

Как это делается?

```

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

<application
    android:name=".SkillSchoolApplication"
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity
        android:name=".ui.activities.SplashActivity"
        android:theme="@style/SplashTheme">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".ui.activities.MainActivity" />
    <activity android:name=".ui.activities.HomeActivity"
        android:parentActivityName=".ui.activities.MainActivity" /> // здесь мы указываем
        // системе, что MainActivity является родителем HomeActivity
</application>
```

Теперь, когда мы будем нажимать на стрелку внутри панели инструментов HomeActivity, это приведет обратно к родительской активности.

Java-код

Здесь приведен соответствующий Java-код, необходимый для этой функциональности:

```
public class HomeActivity extends AppCompatActivity {
    @BindView(R.id.toolbar)
    Toolbar toolbar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_home);
        ButterKnife.bind(this);
        // Поскольку используем пользовательскую панель инструментов, то устанавливаем
        // ссылку на эту панель в Actionbar. Если вы не используете пользовательскую
        // панель, то можете просто оставить это и перейти к следующей строке
        setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true); // при этом на панели
        // инструментов будет отображаться стрелка назад.
    }
}
```

Если вы запустите этот код, то увидите, что при нажатии кнопки «назад» он возвращает вас в `MainActivity`. Для более глубокого понимания навигации вверх можно почитать документацию: <https://developer.android.com/training/implementing-navigation/ancestral.html>.

Вы можете более точно настроить это поведение в зависимости от своих потребностей при помощи следующего переопределения:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        // Реагирование на кнопку Up/Home панели действий
        case android.R.id.home:
            NavUtils.navigateUpFromSameTask(this); // Здесь вы напишете свою логику для
            // обработки навигации вверх
            return true;
    }
    return super.onOptionsItemSelected(item);
}
```

Простой хак

Рассмотрим простой хак, который в основном используется для перехода к родительской активности, если родитель находится в обратном стеке. Вызываем `onBackPressed()`, если `id` равен `android.R.id.home`:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    switch (id) {
        case android.R.id.home:
            onBackPressed();
            return true;
    }
    return super.onOptionsItemSelected(item);
}
```

34.7. Очистка стека текущих активностей и запуск новой активности

Если необходимо очистить текущий стек активностей и запустить новую активность (например, выйти из приложения и запустить активность входа в него), то следует выполнить (при целевом API ≥ 16) вызов функции `finishAffinity()` из активности.

Глава 35. Распознавание активности

Распознавание активности – это определение физической активности пользователя для выполнения определенных действий с устройством, например, набор очков при обнаружении движения, отключение wifi при неподвижном телефоне или установка максимальной громкости звонка при ходьбе.

35.1. Google Play ActivityRecognitionAPI

Это простой пример использования `ActivityRecognitionApi` сервиса GooglePlay. Несмотря на то, что это отличная библиотека, она не работает на устройствах, на которых не установлены сервисы Google Play.

Документация по API `ActivityRecognition`: <https://developers.google.com/android/reference/com/google/android/gms/location/ActivityRecognitionApi>.

Манифест

```
<!-- Это необходимо для использования функции распознавания активности! -->
<uses-permission android:name="com.google.android.gms.permission.ACTIVITY_RECOGNITION" />

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

    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <receiver android:name=".ActivityReceiver" />
</application>
```

MainActivity.java

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

    private GoogleApiClient apiClient;
```

```
private LocalBroadcastManager localBroadcastManager;
private BroadcastReceiver localActivityReceiver;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    apiClient = new GoogleApiClient.Builder(this)
        .addApi(ActivityRecognition.API)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .build();

    // Это просто получение намерения активности из класса ActivityReceiver
    localBroadcastManager = LocalBroadcastManager.getInstance(this);
    localActivityReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            ActivityRecognitionResult recognitionResult =
                ActivityRecognitionResult.extractResult(intent);
            TextView textView = (TextView) findViewById(R.id.activityText);

            // Это для получения имени активности. Используйте на свой страх и риск
            textView.setText(DetectedActivity.zzkf(recognitionResult.getMostProbableActivity().
                getType()));
        }
    };
}

@Override
protected void onResume() {
    super.onResume();
    // Регистрация приемника вещания
    localBroadcastManager.registerReceiver(localActivityReceiver, new
        IntentFilter("activity"));

    // Подключение клиента google api
    apiClient.connect();
}

@Override
protected void onPause() {
    super.onPause();
    // Снять регистрацию для распознавания активности
    ActivityRecognition.ActivityRecognitionApi.removeActivityUpdates(apiClient,
        PendingIntent.getBroadcast(this, 0, new Intent(this, ActivityReceiver.class),
        PendingIntent.FLAG_UPDATE_CURRENT));

    // Отключение api клиента
    apiClient.disconnect();
}

// Снять с регистрации локальный приемник
localBroadcastManager.unregisterReceiver(localActivityReceiver);
}

@Override
```

```

public void onConnected(@Nullable Bundle bundle) {
    //Регистрация для распознавания активности происходит только в том случае,
    //если подключился клиент google api
    ActivityRecognition.ActivityRecognitionApi.requestActivityUpdates(apiClient,
        0, PendingIntent.getBroadcast(this, 0, new Intent(this, ActivityReceiver.
        class), PendingIntent.FLAG_UPDATE_CURRENT));
}

@Override
public void onConnectionSuspended(int i) {}

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

```

ActivityReceiver

```

public class ActivityReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) { LocalBroadcastManager.
        getInstance(context).sendBroadcast(intent.setAction("activity"));
    }
}

```

35.2. Распознавание активности PathSense

PathSense (<http://www.pathsense.com/>) – еще одна хорошая библиотека для устройств без Google Play Services, в ней есть собственная модель распознавания активности, но она требует от разработчиков регистрации на сайте <http://developer.pathsense.com> для получения ключа API и идентификатора клиента.

Манифест

```

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

    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <receiver android:name=".ActivityReceiver" />

<!-- Их необходимо получить на сайте http://developer.pathsense.com-->

```

```

<meta-data
    android:name="com.pathsense.android.sdk.CLIENT_ID"
    android:value="YOUR_CLIENT_ID" />
<meta-data
    android:name="com.pathsense.android.sdk.API_KEY"
    android:value="YOUR_API_KEY" />
</application>

```

MainActivity.java

```

public class MainActivity extends AppCompatActivity {
    private PathsenseLocationProviderApi pathsenseLocationProviderApi;
    private LocalBroadcastManager localBroadcastManager;
    private BroadcastReceiver localActivityReceiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        pathsenseLocationProviderApi = PathsenseLocationProviderApi.getInstance(this);

        // Это просто получение намерения активности из класса ActivityReceiver
        localBroadcastManager = LocalBroadcastManager.getInstance(this);
        localActivityReceiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                // Объект detectedActivities передается как сериализуемый
                PathsenseDetectedActivities detectedActivities =
                    (PathsenseDetectedActivities) intent.getSerializableExtra("ps");
                TextView textView = (TextView) findViewById(R.id.activityText);

                textView.setText(detectedActivities.getMostProbableActivity().getDetectedActivity().name());
            }
        };
    }

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

        // Регистрация приемника местного вещания
        localBroadcastManager.registerReceiver(localActivityReceiver, new
            IntentFilter("activity"));

        // Это дает обновление каждый раз при получении, даже если обновление было
        // таким же, как и предыдущее
        pathsenseLocationProviderApi.requestActivityUpdates(ActivityReceiver.class);

        // Это дает обновления только при изменении (например, ON_FOOT -> IN_VEHICLE)
        // pathsenseLocationProviderApi.requestActivityChanges(ActivityReceiver.class);
    }

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

        pathsenseLocationProviderApi.removeActivityUpdates();
    }
}

```

```

    // pathsenseLocationProviderApi.removeActivityChanges();

    // Снять с регистрации локальный приемник
    localBroadcastManager.unregisterReceiver(localActivityReceiver);
}
}

```

ActivityReceiver.java

```

// Не обязательно использовать Pathsense broadcastreceiver, но лучше сделать это,
// и просто передать результат по мере необходимости в другой класс.
public class ActivityReceiver extends PathsenseActivityRecognitionReceiver {

    @Override
    protected void onDetectedActivities(Context context, PathsenseDetectedActivities
    pathsenseDetectedActivities) {
        Intent intent = new Intent("activity").putExtra("ps",
        pathsenseDetectedActivities);
        LocalBroadcastManager.getInstance(context).sendBroadcast(intent);
    }
}

```

Глава 36. Раздельный экран и многоэкранные действия

Установите этот атрибут в манифесте или элементе для включения или отключения многооконного отображения:

```
android:resizeableActivity=["true" | "false"].
```

Если этот атрибут имеет значение `true`, то активность может быть запущена в режимах `split-screen` и `freeform`. Если атрибут имеет значение `false`, то активность не поддерживает многооконный режим, и если пользователь попытается запустить активность в многооконном режиме, то она займет весь экран.

Если ваше приложение нацелено на уровень API 24, но вы не указали значение для этого атрибута, то по умолчанию значение атрибута равно `true`.

В следующем коде показано, как задать размер и расположение активности по умолчанию, а также ее минимальный размер, если активность отображается в режиме `freeform`:

```

<--Это значения по умолчанию, предложенные google.-->
<activity android:name=".MyActivity">
<layout android:defaultHeight="500dp"
       android:defaultWidth="600dp"
       android:gravity="top|end"
       android:minHeight="450dp"
       android:minWidth="300dp" />
</activity>

```

Отключенные функции в многооконном режиме

Некоторые функции отключаются или игнорируются, когда устройство находится в многооконном режиме, поскольку они не имеют смысла для активности, которая может делить экран устройства с другими действиями или приложениями. Примеры:

1. Некоторые опции настройки пользовательского интерфейса системы отключаются; например, приложения не могут скрыть строку состояния, если они не работают в полноэкранном режиме.
2. Система игнорирует изменения атрибута android:screenOrientation.

Глава 37. Материальный дизайн

Материальный дизайн (Material Design), также известный как «материальное оформление» – это всеобъемлющее руководство по визуальному, динамическому и интерактивному дизайну на разных платформах и устройствах.

37.1. Добавление панели инструментов

Панель инструментов (Toolbar) – это обобщение панели ActionBar для использования в макетах приложений. Если ActionBar традиционно является частью непрозрачного декора окна Activity, управляемого фреймворком, то Toolbar может быть размещен на любом произвольном уровне вложенности в иерархии представлений. Для ее добавления необходимо выполнить следующие действия:

1. Убедитесь, что в файл build.gradle вашего модуля (например, приложения) в разделе dependencies добавлена следующая зависимость:

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

2. Установите для своего приложения тему, в которой отсутствует ActionBar. Для этого отредактируйте файл styles.xml в разделе res/values и установите тему Theme.AppCompat.

В данном примере мы используем Theme.AppCompat.NoActionBar, в качестве родителя вашей AppTheme:

```
<style name="AppTheme" parent="Theme.AppCompat.NoActionBar">
    <item name="colorPrimary">@color/primary</item>
    <item name="colorPrimaryDark">@color/primaryDark</item>
    <item name="colorAccent">@color/accent</item>
</style>
```

Можно также использовать Theme.AppCompat.Light.NoActionBar или Theme.AppCompat.DayNight.NoActionBar либо любую другую тему, которая по своей сути не имеет ActionBar.

3. Добавьте панель инструментов в макет активности:

```
<android.support.v7.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:elevation="4dp" />
```

Под панелью инструментов можно добавить остальные элементы макета.

4. В своей Activity установите панель инструментов в качестве ActionBar для этой Activity. Если вы используете библиотеку appcompat (<https://developer.android.com/topic/libraries/support-library/packages.html#v7-appcompat>) и AppCompatActivity, то следует применить метод setSupportActionBar():

```
@Override
protected void onCreate(Bundle savedInstanceState) {
```

```

super.onCreate(savedInstanceState); setContentView(R.layout.activity_main);

final Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
setSupportActionBar(toolbar);

//...
}

```

После выполнения описанных выше действий можно использовать метод `getSupportActionBar()` для работы с панелью инструментов, установленной в качестве `ActionBar`.

Например, можно задать заголовок, как показано ниже:

```
getSupportActionBar().setTitle("Заголовок активности");
```

Например, можно также задать цвет заголовка и фона, как показано ниже:

```

CharSequence title = "Your App Name";
SpannableString s = new SpannableString(title);
s.setSpan(new ForegroundColorSpan(Color.RED), 0, title.length(), Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
getSupportActionBar().setTitle(s);
getSupportActionBar().setBackgroundDrawable(new ColorDrawable(Color.argb(128, 0, 0, 0)));

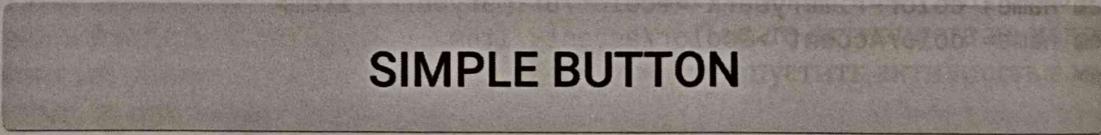
```

37.2. Кнопки, стилизованные под материальный дизайн

Библиотека поддержки AppCompat (<https://developer.android.com/topic/libraries/support-library/features.html#v7-appcompat>) определяет несколько полезных стилей для кнопок, каждый из которых расширяет базовый стиль `Widget.AppCompat.Button`, который по умолчанию применяется ко всем кнопкам, если вы используете тему AppCompat. Этот стиль помогает обеспечить одинаковый вид всех кнопок по умолчанию в соответствии со спецификацией Material Design (<https://material.io/guidelines/components/buttons.html>).

В данном случае акцентным цветом является розовый.

1. Простая кнопка (Simple Button): `@style/Widget.AppCompat.Button`



SIMPLE BUTTON

```
<Button
    style="@style/Widget.AppCompat.Button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="16dp"
    android:text="@string/simple_button"/>
```

2. Цветная кнопка (Colored Button): `@style/Widget.AppCompat.Button.Colored`

Стиль `Widget.AppCompat.Button.Colored` расширяет стиль `Widget.AppCompat.Button` и автоматически применяет акцентный цвет, выбранный в теме приложения.



COLORED BUTTON

```
<Button
    style="@style/Widget.AppCompat.Button.Colored"
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="16dp"
    android:text="@string/colored_button" />
```

Если вы хотите настроить цвет фона без изменения цвета акцента в *основной теме*, вы можете создать *собственную тему* (расширяющую тему `ThemeOverlay`) для кнопки и присвоить ее атрибуту `android:theme` кнопки:

```
<Button
    style="@style/Widget.AppCompat.Button.Colored"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="16dp"
    android:theme="@style/MyButtonTheme" />
```

Определите тему в файле `res/values/themes.xml`:

```
<style name="MyButtonTheme" parent="ThemeOverlay.AppCompat.Light">
    <item name="colorAccent">@color/my_color</item>
</style>
```

3. Кнопка без полей (Borderless Button): `@style/Widget.AppCompat.Button.Borderless`

BORDERLESS BUTTON

```
<Button
    style="@style/Widget.AppCompat.Button.Borderless"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="16dp"
    android:text="@string/borderless_button" />
```

4. Цветная кнопка без полей (Borderless.Colored Button): `@style/Widget.AppCompat.Button.Borderless.Colored`

BORDERLESS COLORED BUTTON

```
<Button
    style="@style/Widget.AppCompat.Button.Borderless.Colored"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="16dp"
    android:text="@string/borderless_colored_button" />
```

37.3. Добавление плавающей кнопки действия (FloatingActionButton, FAB)

В материальном дизайне плавающая кнопка действия (<https://material.google.com/components/buttons-floating-action-button.html>) представляет собой основное действие в активности.

Они отличаются круговой иконкой, находящейся над пользовательским интерфейсом, и имеют поведение движения, включающее морфинг, запуск и перемещение точки привязки.

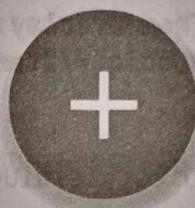
Убедитесь, что в файл build.gradle вашего приложения в разделе зависимостей добавлена следующая зависимость:

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

Теперь добавьте FloatingActionButton в файл макета:

```
<android.support.design.widget.FloatingActionButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="16dp"
    android:src="@drawable/some_icon"/>
```

...где атрибут src ссылается на иконку, которая должна использоваться для плавающего действия. Результат должен выглядеть примерно так:



По умолчанию цвет фона кнопки FloatingActionButton будет соответствовать цвету акцента темы. Также обратите внимание, что для корректной работы кнопки FloatingActionButton требуется наличие поля вокруг нее. Рекомендуемое поле снизу составляет 16dp для телефонов и 24dp для планшетов.

Ниже приведены свойства, которые можно использовать для дальнейшей настройки FloatingActionButton (при условии, что xmlns:app="http://schemas.android.com/apk/res-auto объявляется как пространство имен в верхней части вашего макета):

- app:fabSize: Можно установить значение normal или mini для переключения между нормальным или уменьшенным размером.
- app:rippleColor: Устанавливает цвет эффекта пульсации вашей кнопки FloatingActionButton. Может быть цветовым ресурсом или шестнадцатеричной строкой.
- app:elevation: Может быть строкой, целым числом, логическим значением, значением цвета, значением с плавающей точкой, значением размерности.
- app:useCompatPadding: Включает использование совместимых подкладок. Может быть булевым значением, например true или false. Установите значение true, чтобы использовать совместимые подкладки на API 21 и более поздних версиях, чтобы сохранить согласованный вид с более старыми уровнями API.

37.4. RippleDrawable

Сенсорный эффект Ripple появился в Android 5.0 (уровень API 21) вместе с материальным дизайном, а анимация реализована новым классом RippleDrawable.

Это Drawable-объект, отображающий эффект пульсации в ответ на изменение состояния. Положение привязки пульсации для данного состояния может быть задано вызовом setHotspot(float x, float y) с соответствующим идентификатором атрибута состояния.

В общем случае эффект пульсации для **обычных кнопок** работает по умолчанию в API 21 и выше, а для других подобных объектов касания его можно получить, указав:

```
        android:background="?android:attr/selectableItemBackground">
```

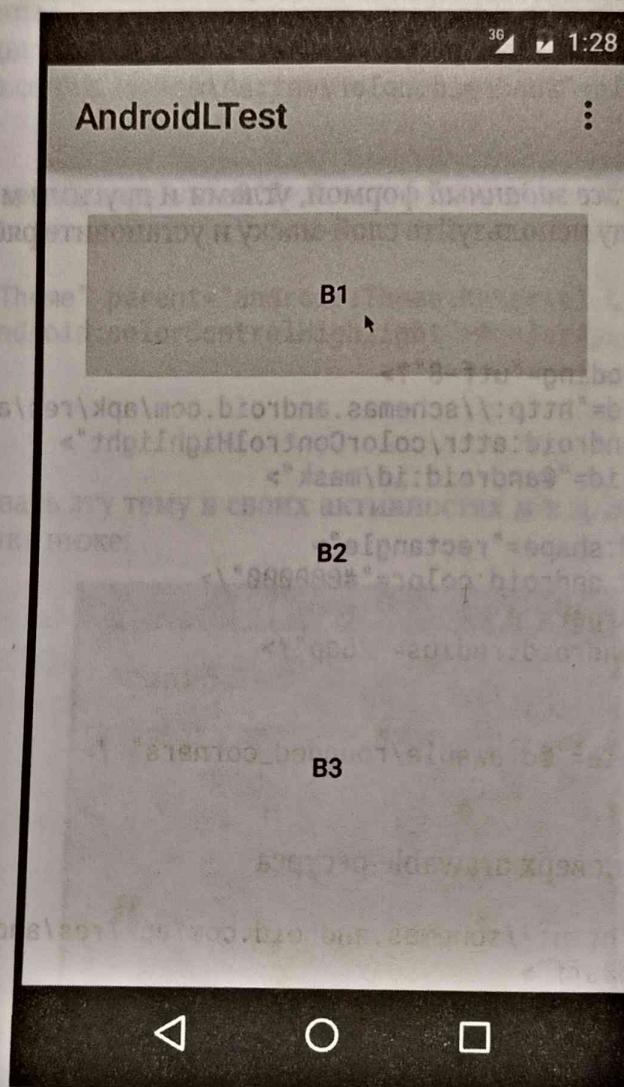
...для пульсаций, содержащихся в представлении:

```
android:background="?android:attr/selectableItemBackgroundBorderless"
```

...или для пульсаций, выходящих за границы представления.

Например, на изображении ниже

- В1 – это кнопка, не имеющая фона
- В2 задается с помощью `android:background="android:attr/selectableItemBackground"`
- В3 устанавливается с помощью `android:background="android:attr/selectableItemBackgroundBorderless"`



(Изображение взято с <http://blog.csdn.net/a396901990/article/details/40187203>)

Того же самого можно добиться в коде, используя:

```
int[] attrs = new int[]{R.attr.selectableItemBackground};  
TypedArray typedArray = getActivity().obtainStyledAttributes(attrs);  
int backgroundResource = typedArray.getResourceId(0, 0);  
  
myView.setBackgroundResource(backgroundResource);
```

Пульсации также могут быть добавлены к представлению с помощью атрибута `android:foreground` аналогично тому, как это было описано выше. Как следует из названия, в случае добавления ряби на передний план она будет отображаться над любым представлением, к которому она добавлена (например, `ImageView`, `LinearLayout`, содержащий несколько представлений, и т. д.).

Если вы хотите настроить эффект пульсации в представлении, необходимо создать XML-файл внутри каталога drawable.

Приведем несколько примеров:

Пример 1: Неограниченная пульсация

```
<ripple xmlns:android="http://schemas.android.com/apk/res/android"
    android:color="#ffff0000" />
```

Пример 2: Пульсация с использованием маски и цвета фона

```
<ripple android:color="#7777777"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/mask"
        android:drawable="#ffff00" />
    <item android:drawable="@+color/white" />
</ripple>
```

Если у вида есть фон, уже заданный формой, углами и другими метками, то для добавления пульсации к этому виду используйте слой-маску и установите рябь в качестве фона вида.

Пример:

```
<?xml version="1.0" encoding="utf-8"?>
<ripple xmlns:android="http://schemas.android.com/apk/res/android"
    android:color="?android:attr/colorControlHighlight">
    <item android:id="@+id/mask">
        <shape
            android:shape="rectangle">
            solid android:color="#000000" />
        <corners
            android:radius="25dp" />
    </shape>
    </item>
    <item android:drawable="@drawable/rounded_corners" />
</ripple>
```

Пример 3: Пульсация поверх drawable-ресурса

```
<ripple xmlns:android="http://schemas.android.com/apk/res/android"
    android:color="#ff0000ff">
    <item android:drawable="@drawable/my_drawable" />
</ripple>
```

Использование: чтобы прикрепить свой XML-файл пульсации к любому представлению, установите его в качестве фона следующим образом (при условии, что ваш файл пульсации имеет имя my_ripple.xml):

```
<View
    android:id="@+id/myViewId"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/my_ripple" />
```

Селектор:

Для целевой версии 21 или выше вместо селекторов списка состояний цветов можно использовать селектор ripple drawable (селектор ripple можно поместить в папку drawable-v21):

```
<!-- /drawable/button.xml: -->
<selector xmlns:android="http://schemas.android.com/apk/res/android">
```

```
<item android:state_pressed="true" android:drawable="@drawable/button_pressed"/>
<item android:drawable="@drawable/button_normal"/>
</selector>

<!--/drawable-v21/button.xml-->
<?xml version="1.0" encoding="utf-8"?>
<ripple xmlns:android="http://schemas.android.com/apk/res/android"
    android:color="?android:colorControlHighlight">
    <item android:drawable="@drawable/button_normal" />
</ripple>
```

В этом случае цвет состояния представления по умолчанию будет белым, а в нажатом состоянии будет отображаться отрисовываемая пульсация.

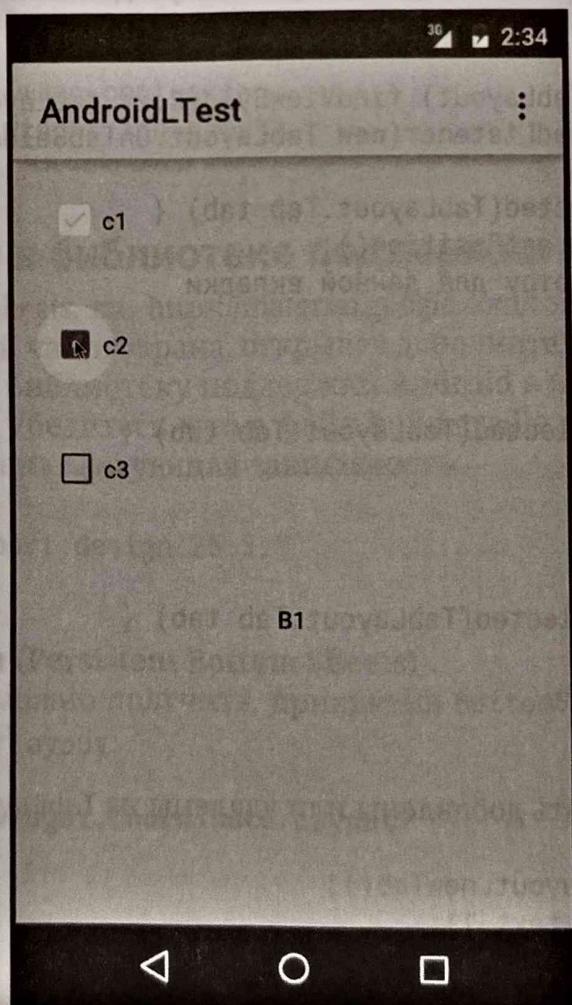
Обратите внимание: при использовании ?android:colorControlHighlight пульсация будет иметь тот же цвет, что и встроенные пульсации в вашем приложении. Чтобы изменить только цвет пульсации, можно настроить цвет android:colorControlHighlight в теме следующим образом:

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

    <style name="AppTheme" parent="android:Theme.Material.Light.DarkActionBar">
        <item name="android:colorControlHighlight">@color/your_custom_color</item>
    </style>

</resources>
```

...а затем использовать эту тему в своих активностях и т. д. Эффект будет выглядеть так, как показано на рисунке ниже:



37.5. Добавление TabLayout

TabLayout обеспечивает горизонтальный макет для отображения вкладок и обычно используется в сочетании с ViewPager. Дополнительно см. <https://developer.android.com/reference/android/support/design/widget/TabLayout.html>.

Убедитесь, что в файл build.gradle вашего приложения в разделе зависимостей добавлена следующая зависимость:

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

Теперь вы можете добавлять элементы на TabLayout в своем макете, используя класс TabItem. Например:

```
<android.support.design.widget.TabLayout
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:id="@+id/tabLayout">

    <android.support.design.widget.TabItem
        android:text="@string/tab_text_1"
        android:icon="@drawable/ic_tab_1"/>

    <android.support.design.widget.TabItem
        android:text="@string/tab_text_2"
        android:icon="@drawable/ic_tab_2"/>

</android.support.design.widget.TabLayout>
```

Добавьте OnTabSelectedListener для получения уведомления о том, что произошел выбор, отмена выбора или повторный выбор вкладки в TabLayout:

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.tabLayout);
tabLayout.addOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
    @Override
    public void onTabSelected(TabLayout.Tab tab) {
        int position = tab.getPosition();
        // Переход к просмотру для данной вкладки
    }

    @Override
    public void onTabUnselected(TabLayout.Tab tab) {
    }

    @Override
    public void onTabReselected(TabLayout.Tab tab) {
    }
});
```

Вкладки также могут быть добавлены или удалены из TabLayout программно:

```
TabLayout.Tab tab = tabLayout.newTab();
tab.setText(R.string.tab_text_1);
tab.setIcon(R.drawable.ic_tab_1);
tabLayout.addTab(tab);

tabLayout.removeTab(tab);
```

```
tabLayout.removeTabAt(0);
tabLayout.removeAllTabs();
```

TabLayout имеет два режима – фиксированный (fixed) и прокручиваемый (scrollable).

```
tabLayout.setTabMode(TabLayout.MODE_FIXED);
tabLayout.setTabMode(TabLayout.MODE_SCROLLABLE);
```

Они также могут быть применены в XML:

```
<android.support.design.widget.TabLayout
    android:id="@+id/tabLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:tabMode="fixed|scrollable" />
```

Примечание: режимы TabLayout являются взаимоисключающими, то есть одновременно может быть активен только один из них. Цвет индикатора вкладки – это цвет акцента, заданный в вашей теме материального дизайна. Вы можете переопределить этот цвет, определив пользовательский стиль в файле styles.xml и применив его к вашему TabLayout:

```
<style name="MyCustomTabLayoutStyle" parent="Widget.Design.TabLayout">
    <item name="tabIndicatorColor">@color/your_color</item>
</style>
```

Затем можно применить стиль к представлению с помощью:

```
<android.support.design.widget.TabLayout
    android:id="@+id/tabs"
    style="@style/MyCustomTabLayoutStyle"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
</android.support.design.widget.TabLayout>
```

37.6. Нижние листы в библиотеке поддержки проектирования

Нижние листы (Bottom sheets, см. <https://material.google.com/components/bottom-sheets.html>) сдвигаются вверх из нижней части экрана, открывая дополнительные материалы.

Они были добавлены в библиотеку поддержки Android в версии v25.1.0 и поддерживаются выше всех версий. Убедитесь, что в файл build.gradle вашего приложения в разделе зависимостей добавлена следующая зависимость:

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

Стойкие нижние листы (Persistent Bottom Sheets)

Стойкий нижний лист можно получить, прикрепив BottomSheetBehavior к дочернему представлению CoordinatorLayout:

```
<android.support.design.widget.CoordinatorLayout>
```

```
<!-- ... -->
```

```
<LinearLayout
```

```
    android:id="@+id/bottom_sheet"
    android:elevation="4dp"
```

```

    android:minHeight="120dp"
    app:behavior_peekHeight="120dp"
    ...
    app:layout_behavior="android.support.design.widget.BottomSheetBehavior">

        <!-- ..... -->

    </LinearLayout>

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

```

Затем в коде можно создать ссылку с помощью:

```

// Представление с поведением BottomSheetBehavior
View bottomSheet = coordinatorLayout.findViewById(R.id.bottom_sheet);
BottomSheetBehavior mBottomSheetBehavior = BottomSheetBehavior.from(bottomSheet);

Установить состояние BottomSheetBehavior можно с помощью метода setState():

mBottomSheetBehavior.setState(BottomSheetBehavior.STATE_EXPANDED);

```

Вы можете использовать одно из этих состояний:

- STATE_COLLAPSED: это свернутое состояние используется по умолчанию и показывает только часть макета снизу. Высоту можно регулировать с помощью атрибута `app:behavior_peekHeight` (по умолчанию 0)
- STATE_EXPANDED: полностью развернутое состояние нижнего листа, при котором либо виден весь нижний лист (если его высота меньше высоты содержащего его CoordinatorLayout), либо заполнен весь CoordinatorLayout
- STATE_HIDDEN: по умолчанию отключен (включается с помощью атрибута `app:behavior_hideable`), включение этого параметра позволяет пользователям провести пальцем вниз по нижнему листу, чтобы полностью его скрыть

Далее, чтобы открывать или закрывать лист BottomSheet по щелчку на выбранном вами представлении, скажем, кнопке, переключить поведение листа и обновить вид можно следующим образом:

```

mButton = (Button) findViewById(R.id.button_2);
//По нажатию кнопки мы отслеживаем состояние листа
mButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (mBottomSheetBehavior.getState() == BottomSheetBehavior.STATE_EXPANDED) {
            //Если развернуто, то свернуть (настройка в режиме Peek).
            mBottomSheetBehavior.setState(BottomSheetBehavior.STATE_COLLAPSED);
            mButton.setText(R.string.button2_hide);
        } else if (mBottomSheetBehavior.getState() == BottomSheetBehavior.STATE_COLLAPSED) {
            //Если свернуто, то скрыть полностью.
            mBottomSheetBehavior.setState(BottomSheetBehavior.STATE_HIDDEN);
            mButton.setText(R.string.button2);
        } else if (mBottomSheetBehavior.getState() == BottomSheetBehavior.STATE_HIDDEN) {
            //Если скрыт, то свернуть или развернуть, в зависимости от необходимости.
            mBottomSheetBehavior.setState(BottomSheetBehavior.STATE_EXPANDED);
            mButton.setText(R.string.button2_peek);
        }
    }
});

```

Но в поведении BottomSheet также есть функция, при которой пользователь может взаимодействовать с листом, проводя по нему вверх или вниз движением DRAG. В этом случае мы не сможем обновить зависимое представление (как в случае с кнопкой выше), если состояние листа изменилось. В этом случае появляется необходимость получать обратные вызовы об изменении состояния, поэтому можно добавить функцию BottomSheetCallback для прослушивания событий пролистывания пользователем листа:

```
mBottomSheetBehavior.setBottomSheetCallback(new BottomSheetCallback() {
    @Override
    public void onStateChanged(@NonNull View bottomSheet, int newState) {
        // Реакция на изменение состояния и уведомление представлений о текущем
        // состоянии
    }
    @Override
    public void onSlide(@NonNull View bottomSheet, float slideOffset) {
        // Реакция на события перетаскивания и анимирования представлений или
        // прозрачности зависимых представлений
    }
});
```

А если вы хотите, чтобы нижний лист был виден только в режимах COLLAPSED и EXPANDED, и никогда в режиме HIDE, используйте:

```
mBottomSheetBehavior2.setHideable(false);
```

Диалоговый фрагмент нижнего листа

Вместо View на нижнем листе можно отобразить диалоговый фрагмент нижнего листа (BottomSheetDialogFragment). Для этого сначала необходимо создать новый класс, расширяющий этот диалоговый фрагмент.

В методе setupDialog() можно «раздуть» новый файл макета и получить поведение BottomSheetBehavior контейнерного представления в вашей Activity. Получив такое поведение, можно создать и связать с ним BottomSheetCallback для отключения фрагмента при скрытии листа.

```
public class BottomSheetDialogFragmentExample extends BottomSheetDialogFragment {
    private BottomSheetBehavior.BottomSheetCallback mBottomSheetBehaviorCallback =
        new BottomSheetBehavior.BottomSheetCallback() {

            @Override
            public void onStateChanged(@NonNull View bottomSheet, int newState) {
                if (newState == BottomSheetBehavior.STATE_HIDDEN) {
                    dismiss();
                }
            }

            @Override
            public void onSlide(@NonNull View bottomSheet, float slideOffset) {
            }
        };

        @Override
        public void setupDialog(Dialog dialog, int style) {
            super.setupDialog(dialog, style);
            View contentView = View.inflate(getContext(), R.layout.fragment_bottom_sheet,
                null);
        }
};
```

```

        dialog.setContentView(contentView);

        CoordinatorLayout.LayoutParams params = (CoordinatorLayout.LayoutParams)
        ((View) contentView.getParent()).getLayoutParams();
        CoordinatorLayout.Behavior behavior = params.getBehavior();

        if( behavior != null && behavior instanceof BottomSheetBehavior ) {
            ((BottomSheetBehavior) behavior).
            setBottomSheetCallback(mBottomSheetBehaviorCallback);
        }
    }
}

```

Наконец, можно вызвать функцию `show()` для экземпляра фрагмента, чтобы отобразить его на нижнем листе.

```

BottomSheetDialogFragment bottomSheetDialogFragment = new
BottomSheetDialogFragmentExample();
bottomSheetDialogFragment.show(getSupportFragmentManager(),
bottomSheetDialogFragment.getTag());

```

37.7. Применение темы AppCompat

Библиотека поддержки AppCompat предоставляет темы для создания приложений с использованием спецификации Material Design – «материального оформления» (см. <https://material.google.com/>). Тема с родителем `Theme.AppCompat` также необходима для того, чтобы активность расширяла `AppCompatActivity`.

Первым шагом является настройка цветовой палитры темы для автоматической раскраски приложения. В файле `res/styles.xml` приложения можно определить:

```

<!-- наследование от темы AppCompat -->
<style name="AppTheme" parent="Theme.AppCompat">

    <!-- брендовый цвет вашего приложения для панели приложений -->
    <item name="colorPrimary">#2196f3</item>

    <!-- более темный вариант для строки состояния и контекстных панелей приложений -->
    <item name="colorPrimaryDark">#1976d2</item>

    <!-- элементы управления пользовательским интерфейсом темы, такие как флагки и текстовые поля -->
    <item name="colorAccent">#f44336</item>
</style>

```

Вместо `Theme.AppCompat`, которая имеет темный фон, можно также использовать темы `Theme.AppCompat.Light` или `Theme.AppCompat.Light.DarkActionBar`.

Вы можете настроить тему, используя собственные цвета. Хорошие варианты можно найти в таблице цветов спецификации Material Design (<http://www.google.com/design/spec/style/color.html#>) и в палитре Material Palette (<https://www.materialpalette.com/>). Цвета “500” хорошо подходят для основного (в данном примере – синий 500); для темного выберите “700” того же оттенка, а для акцентного цвета – оттенок другого цвета. Основной цвет используется для панели инструментов приложения и записи о нем на экране обзора (последние приложения), более темный вариант – для оттенка строки состояния, а акцентный – для выделения некоторых элементов управления.

После создания этой темы примените ее к своему приложению в `AndroidManifest.xml`, а также примените тему к любой конкретной активности. Это удобно для применения темы `AppTheme.NoActionBar`, которая позволяет реализовать нестандартные конфигурации панели инструментов.

```
<application android:theme="@style/AppTheme"
    ...
    <activity
        android:name=".MainActivity"
        android:theme="@style/AppTheme" />
</application>
```

Также можно применять темы к отдельным представлениям, используя `android:theme` и тему `ThemeOverlay`.

Например, с панелью инструментов:

```
<android.support.v7.widget.Toolbar
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="?attr/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" />
```

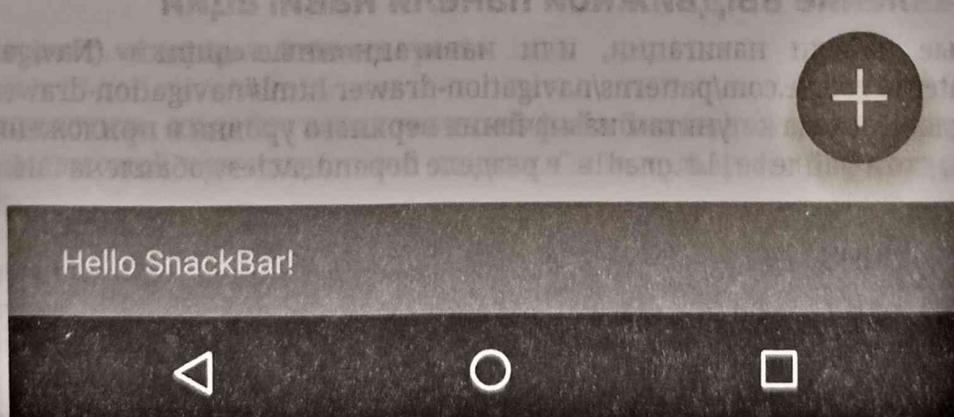
...или кнопкой:

```
<Button
    style="@style/Widget.AppCompat.Button.Colored"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:theme="@style/MyButtonTheme" />
<!-- res/values/themes.xml -->
<style name="MyButtonTheme" parent="ThemeOverlay.AppCompat.Light">
    <item name="colorAccent">@color/my_color</item>
</style>
```

37.8. Добавление всплывающего уведомления (Snackbar)

Одной из главных особенностей материального дизайна (материального оформления) является добавление Snackbar, «закусочной панели» всплывающих уведомлений, которая теоретически заменяет предыдущую систему уведомлений, основанную на Toast-сообщениях. В соответствии с документацией по Android:

Snackbar содержит одну строку текста, непосредственно связанного с выполняемой операцией. Они могут содержать текстовое действие, но не содержать иконок. Тости (Toasts) используются в основном для передачи системных сообщений. Они также отображаются в нижней части экрана, но не могут быть выведены за его пределы.



Тости по-прежнему могут использоваться в Android для отображения сообщений пользователям, однако если вы решили использовать материальный дизайн в своем приложении, рекомендуется использовать snackbars. Вместо того чтобы отображаться в виде наложения (overlay) на экране, Snackbar высекает снизу.

Вот как это делается:

```
Snackbar snackbar = Snackbar
    .make(coordinatorLayout, "Here is your new Snackbar", Snackbar.LENGTH_LONG);
snackbar.show();
```

Что касается длительности показа Snackbar, то здесь есть варианты, аналогичные тем, что предлагает Toast, или же можно задать пользовательскую длительность в миллисекундах:

- LENGTH_SHORT
- LENGTH_LONG
- LENGTH_INDEFINITE
- setDuration() (с версии 22.2.1)

Кроме того, можно добавить динамические функции в ваш Snackbar, например, обратный вызов действия (ActionCallback) или настраиваемый цвет. Однако при настройке следует обратить внимание на рекомендации по дизайну, предлагаемые Android (<https://material.io/guidelines/components/snackbars-toasts.html#>).

Реализация Snackbar имеет одно ограничение. Родительским макетом представления, в котором вы собираетесь реализовать Snackbar, должен быть CoordinatorLayout. Это необходимо для того, чтобы можно было сделать фактическое всплытие снизу.

Вот как определить CoordinatorLayout в файле layout.xml:

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/coordinatorLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    //любые другие виджеты в вашем макете помещаются сюда
</android.support.design.widget.CoordinatorLayout>
```

Затем CoordinatorLayout необходимо определить в методе onCreate вашей активности, а затем использовать при создании самого Snackbar.

Более подробную информацию о Snackbar можно найти в официальной документации (<https://developer.android.com/reference/android/support/design/widget/Snackbar.html>).

37.9. Добавление выдвижной панели навигации

Выдвижные панели навигации, или навигационные ящики (Navigation Drawers, см. <https://material.google.com/patterns/navigation-drawer.html#navigation-drawer-content>), используются для перехода к пунктам назначения верхнего уровня в приложении.

Убедитесь, что в файле build.gradle в разделе dependencies добавлена библиотека design support:

```
dependencies {
    ...
    compile 'com.android.support:design:25.3.1'
}
```

Затем добавьте DrawerLayout и NavigationView в файл ресурсов XML-макета.

DrawerLayout – это своеобразный контейнер, который позволяет NavigationView, собственно выдвижной навигационной панели, выдвигаться из левой или правой части экрана.

Примечание: для мобильных устройств стандартный размер ящика составляет 320dp.

```

<!-- res/layout/activity_main.xml -->
<android.support.v4.widget.DrawerLayout
    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:id="@+id/navigation_drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:openDrawer="start">
    <!-- Для открытия navigation drawer с правой стороны можно использовать "end" -->

    <android.support.design.widget.CoordinatorLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:fitsSystemWindows="true">

        <android.support.design.widget.AppBarLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:theme="@style/AppTheme.AppBarOverlay">

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

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

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

    <android.support.design.widget.NavigationView
        android:id="@+id/navigation_drawer"
        android:layout_width="320dp"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:fitsSystemWindows="true"
        app:headerLayout="@layout/drawer_header"
        app:menu="@menu/navigation_menu" />

</android.support.v4.widget.DrawerLayout>

```

Теперь, если хотите, создайте **заголовочный файл** (header), который будет служить верхней частью выдвижной навигационной панели. Он используется для придания более элегантного вида:

```

<!-- res/layout/drawer_header.xml -->
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="190dp">

    <ImageView
        android:id="@+id/header_image"
        android:layout_width="140dp"
        android:layout_height="120dp" />

```

```

    android:layout_centerInParent="true"
    android:scaleType="centerCrop"
    android:src="@drawable/image" />

<TextView
    android:id="@+id/header_text_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/header_image"
    android:text="User name"
    android:textSize="20sp" />

</RelativeLayout>

```

На него ссылается тег `NavigationView` в атрибуте `app:headerLayout="@layout/drawer_header"`.

Этот `app:headerLayout` автоматически вставляет указанный макет в заголовок. В качестве альтернативы это можно сделать во время выполнения с помощью:

```

// Просмотр навигационного представления
NavigationView navigationView = (NavigationView) findViewById(R.id.navigation_drawer);
// Развернуть представление заголовка во время выполнения программы
View headerLayout = navigationView.inflateHeaderView(R.layout.drawer_header);

```

Чтобы автоматически заполнить выдвижную панель навигации элементами навигации, соответствующими материальному дизайну, создайте файл меню и добавьте элементы по мере необходимости.

Примечание: хотя значки для элементов не являются обязательными, они предлагаются в спецификации материального оформления Material Design (см. <https://material.io/guidelines/components/menus.html#menus-usage>). На них ссылается тег `NavigationView` в атрибуте `app:menu="@menu/navigation_menu"`:

```

<!-- res/menu/menu_drawer.xml -->
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/nav_item_1"
        android:title="Item #1"
        android:icon="@drawable/ic_nav_1" />
    <item
        android:id="@+id/nav_item_2"
        android:title="Item #2"
        android:icon="@drawable/ic_nav_2" />
    <item
        android:id="@+id/nav_item_3"
        android:title="Item #3"
        android:icon="@drawable/ic_nav_3" />
    <item
        android:id="@+id/nav_item_4"
        android:title="Item #4"
        android:icon="@drawable/ic_nav_4" />
</menu>

```

Чтобы разделить элементы на группы, поместите их в `<menu>`, вложенное в другой `<item>` с атрибутом `android:title`, или оберните их тегом `<group>`.

Теперь, когда разметка выполнена, перейдем к коду Activity:

```

// Найти представление навигации
NavigationView navigationView = (NavigationView) findViewById(R.id.navigation_drawer);

```

```

navigationView.setNavigationItemSelectedListener(new NavigationView.OnNavigationItemSelectedListener() {
    @Override
    public boolean onNavigationItemSelected(MenuItem item) {
        // Получение идентификатора элемента для определения действий при клике
        // пользователя
        int itemId = item.getItemId();
        // Реагируем на выбор Navigation Drawer с помощью нового интента
        startActivity(new Intent(this, OtherActivity.class));
        return true;
    }
});

DrawerLayout drawer = (DrawerLayout) findViewById(R.id.navigation_drawer_layout);
// Необходим для автоматического анимирования Navigation Drawer при открытии
// и закрытии
ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(this, drawer, "Open
navigation drawer", "Close navigation drawer");
// Эти две строки не отображаются пользователю, но не забудьте поместить их
// в отдельный файл strings.xml.
drawer.addDrawerListener(toggle);
toggle.syncState();

```

Теперь в представлении заголовка NavigationView можно делать все что угодно:

```

View headerView = navigationView.getHeaderView();
TextView headerTextView = (TextView) headerView.findViewById(R.id.header_text_view);
ImageView headerImageView = (ImageView) headerView.findViewById(R.id.header_image);
// Установка текста заголовка навигации
headerTextView.setText("Имя пользователя");
// Установка изображения заголовка навигации
headerImageView.setImageResource(R.drawable.header_image);

```

Представление заголовка ведет себя как любое другое представление, поэтому после использования функции `findViewById()` и добавления других представлений в файл компоновки вы можете установить в нем свойства любого представления.

37.10. Как использовать TextInputLayout

Убедитесь, что в файл `build.gradle` вашего приложения в разделе зависимостей добавлена следующая зависимость:

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

Отображение подсказки из `EditText` в виде плавающей метки при вводе значения:

```

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

    <android.support.design.widget.TextInputEditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/form_username"/>

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

```

Для отображения пиктограммы глаза пароля с помощью `TextInputLayout` можно воспользоваться следующим кодом:

```

<android.support.design.widget.TextInputLayout
    android:id="@+id/input_layout_current_password"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:passwordToggleEnabled="true">

    <android.support.design.widget.TextInputEditText

        android:id="@+id/current_password"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/current_password"
        android:inputType="textPassword" />

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

```

...где обязательными являются следующие параметры: `app:passwordToggleEnabled="true"` и `android:inputType="textPassword"`. Приложение должно использовать пространство имен `xmlns:app="http://schemas.android.com/apk/res-auto"`.

Глава 38. Ресурсы

38.1. Определение цвета

Цвета обычно хранятся в файле ресурсов с именем `colors.xml` в папке `/res/values/`. Они определяются элементами `<color>`:

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>

    <color name="blackOverlay">#66000000</color>
</resources>

```

Цвета представляются шестнадцатеричными значениями цвета для каждого цветового канала (0 - FF) в одном из форматов:

- #RGB
- #ARGB
- #RRGGBB
- #AARRGGBB

Обозначения:

- A – альфа-канал (значение 0 – полностью прозрачно, значение FF – непрозрачно)
- R – красный канал
- G – зеленый канал
- B – синий канал

Определенные цвета могут быть использованы в XML со следующим синтаксисом: `@color/name_of_the_color`. Например:

```

<RelativeLayout
    android:layout_width="match_parent"

```

```
    android:layout_height="match_parent"
    android:background="@color/blackOverlay">
```

Использование цвета в коде

В данных примерах предполагается, что **this** – ссылка на активность. Может быть использована и Context-ссылка.

```
int color = ContextCompat.getColor(this, R.color.black_overlay);
view.setBackgroundColor(color);
```

В приведенном выше объявлении `colorPrimary`, `colorPrimaryDark` и `colorAccent` используются для определения цветов материального дизайна, которые будут использоваться при определении пользовательской темы Android в `styles.xml`. Они автоматически добавляются при создании нового проекта в Android Studio.

38.2. Уровень прозрачности цвета (альфа)

Приведем таблицу шестнадцатеричных значений непрозрачности:

Alpha(%)	Hex Value
100%	FF
95%	F2
90%	E6
85%	D9
80%	CC
75%	BF
70%	B3
65%	A6
60%	99
55%	8C
50%	80
45%	73
40%	66
35%	59
30%	4D
25%	40
20%	33
15%	26
10%	1A
5%	0D
0%	00

Например, если вы хотите установить 45% на красный цвет:

```
<color name="red_with_alpha_45">#73FF0000</color>
```

Шестнадцатеричное значение для красного цвета – #FF0000, а в префикс можно добавить 73 для 45% непрозрачности – #73FF0000.

38.3. Определение множественного числа строк

Чтобы различать множественное и единственное число строк, можно определить множественное число в файле *strings.xml* и перечислить различные количества, как показано в примере ниже:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals name="hello_people">
        <item quantity="one">Hello to %d person</item>
        <item quantity="other">Hello to %d people</item>
    </plurals>
</resources>
```

Доступ к этому определению можно получить из Java-кода с помощью метода *getQuantityString()* класса *Resources*, как показано в следующем примере:

```
getResources().getQuantityString(R.plurals.hello_people, 3, 3);
```

Здесь первый параметр *R.plurals.hello_people* – это имя ресурса. Второй параметр (в данном примере – 3) используется для выбора правильной строки количества (*quantity*). Третий параметр (в данном примере также 3) – это аргумент формата, который будет использоваться для подстановки спецификатора формата *%d*.

Возможные значения количества (*quantity*), перечисленные в алфавитном порядке:

few	
many	
one	
other	
two	
zero	

Важно отметить, что не все языки поддерживают любое обозначение количества. Например, в китайском языке нет понятия “один предмет”. Неподдерживаемые экземпляры количества (*quantity*) будут отмечены в IDE как предупреждения Lint, но не вызовут ошибок при их использовании.

38.4. Определение строк

Строки обычно хранятся в файле ресурсов *strings.xml*. Они определяются с помощью XML-элемента **<string>**.

Назначение *strings.xml* – обеспечить возможность интернационализации. Для ISO-кода каждого языка можно определить свой *strings.xml*. Таким образом, когда система ищет строку ‘*app_name*’, она сначала проверяет xml-файл, соответствующий текущему языку, и, если он не найден, ищет запись в стандартном файле *strings.xml*. Это означает, что вы можете выбрать локализацию только некоторых строк и не локализовать другие:

```
/res/values/strings.xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Hello World App</string>
```