

Результат:



Этот метод проверяет

public String getLocalizedName(@StringRes int id) {

String name = null;

// ищет строку в ресурсах

date = format(new Date());

String localizedDate = DateUtils.formatDateTime(context, date.

getTime(), DateUtils.FORMAT_SHOW_DATE | DateUtils.FORMAT_SHOW_WEEKDAY);

DateUtils.formatDateTime() автоматически отслеживает правильность формата даты.

Глава 97. Локализованные даты и время

97.1. Пользовательский локализованный формат даты с помощью DateUtils.formatDateTime()

DateUtils.formatDateTime() (см. <https://developer.android.com/reference/android/text/format/DateUtils.html#formatDateTime>) позволяет указать время, и на основе заданных флагов создается локализованная строка времени. Флаги позволяют указать, нужно ли включать определенные элементы (например, день недели).

Date date = new Date(); String localizedDate = DateUtils.formatDateTime(context, date.getTime(), DateUtils.FORMAT_SHOW_DATE | DateUtils.FORMAT_SHOW_WEEKDAY);

formatDateTime() автоматически отслеживает правильность формата даты.

97.2. Стандартное форматирование даты и времени

Форматирование даты:

```
Date date = new Date(); DateFormat df = DateFormat.getDateInstance(DateFormat.MEDIUM);
String localizedDate = df.format(date)
```

Форматирование даты и времени. Дата представлена в формате SHORT, время – в формате LONG:

```
Date date = new Date(); DateFormat df = DateFormat.getDateTimeInstance(DateFormat.SHORT,
DateFormat.LONG); String localizedDate = df.format(date)
```

97.3. Полностью настраиваемые дата и время

Используйте:

```
Date date = new Date(); df = new SimpleDateFormat("HH:mm", Locale.US); String
localizedDate = df.format(date)
```

Часто используемые шаблоны:

- HH: час (0-23)
- hh: час (1-12)
- a: Маркер AM/PM (до и после полудня)
- mm: минута (0-59)
- ss: секунда
- dd: день в месяце (1-31)
- MM: месяц
- yyyy: год

Глава 98. Возможности для работы со временем

98.1. Проверка в течение периода

Этот пример поможет проверить, находится ли заданное время в пределах определенного периода.

Чтобы проверить, какое время сегодня, мы можем использовать класс DateUtils:

```
boolean isToday = DateUtils.isToday(timeInMillis);
```

Для проверки, входит ли время в неделю:

```
private static boolean isWithinWeek(final long millis) {
    return System.currentTimeMillis() - millis <= (DateUtils.WEEK_IN_MILLIS -
    DateUtils.DAY_IN_MILLIS);
}
```

Для проверки, входит ли время в год:

```
private static boolean isWithinYear(final long millis) {
    return System.currentTimeMillis() - millis <= DateUtils.YEAR_IN_MILLIS;
}
```

Для проверки времени в пределах нескольких дней, включая сегодняшний:

```
public static boolean isWithinDay(long timeInMillis, int day) {
    long diff = System.currentTimeMillis() - timeInMillis;
    float dayCount = (float) (diff / DateUtils.DAY_IN_MILLIS);
    return dayCount < day;
}
```

Примечание: DateUtils – это android.text.format.DateUtils.

98.2. Преобразование формата даты в миллисекунды

Для преобразования даты в формате dd/MM/yyy в миллисекунды необходимо вызвать следующую функцию с данными в виде String:

```
public long getMilliFromDate(String dateFormat) {
    Date date = new Date();
    SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy");
    try {
        date = formatter.parse(dateFormat);
    } catch (ParseException e) {
        e.printStackTrace();
    }
    System.out.println("Сегодня " + date);
    return date.getTime();
}
```

Этот метод преобразует миллисекунды в дату в формате Timestamp:

```
public String getTimeStamp(long timeinMillies) {
    String date = null;
    SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    // модификация формата
    date = formatter.format(new Date(timeinMillies));
    System.out.println("Сегодня " + date);

    return date;
}
```

Этот метод преобразует заданные день, месяц и год в миллисекунды. Это будет очень полезно при использовании Timerpicker или Datepicker.

```
public static long getTimeInMillis(int day, int month, int year) {
    Calendar calendar = Calendar.getInstance();
    calendar.set(year, month, day);
    return calendar.getTimeInMillis();
}
```

Он возвращает миллисекунды от даты:

```
public static String getNormalDate(long timeInMillies) {
```

```

String date = null;
SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy");
date = formatter.format(timeInMillies);
System.out.println("Сегодня " + date);
return date;
}

```

Будет возвращена текущая дата:

```

public static String getCurrentDate() {
    Calendar c = Calendar.getInstance();
    System.out.println("Текущее время => " + c.getTime());
    SimpleDateFormat df = new SimpleDateFormat("dd/MM/yyyy");
    String formattedDate = df.format(c.getTime());
    return formattedDate;
}

```

Примечание: Java обеспечивает поддержку Date Pattern (см. <https://developer.android.com/reference/java/text/SimpleDateFormat.html#number>).

98.3. Метод GetCurrentRealTime

Этот метод позволяет вычислить текущее время устройства и сложить или вычесть разницу между реальным временем и временем устройства:

```

public static Calendar getCurrentRealTime() {
    long bootTime = networkTime - SystemClock.elapsedRealtime();
    Calendar calInstance = Calendar.getInstance(); calInstance.
    setTimeZone(getUTCTimeZone());
    long currentDeviceTime = bootTime + SystemClock.elapsedRealtime();
    calInstance.setTimeInMillis(currentDeviceTime);
    return calInstance;
}

```

Получить часовой пояс, основанный на UTC:

```

public static TimeZone getUTCTimeZone() {
    return TimeZone.getTimeZone("GMT");
}

```

Глава 99. Встроенные покупки

99.1. Расходуемые встроенные покупки

Расходуемые (Consumable) управляемые товары в приложении – то, что можно купить несколько раз, например, внутриигровая валюта, игровые жизни, бонусы и т. д.

В данном примере мы собираемся реализовать 4 различных расходуемых управляемых товара “item1”, “item2”, “item3”, “item4”.

Составим список необходимых шагов:

1. Добавим библиотеку встроенных покупок в свой проект (файл AIDL).
2. Добавим необходимое разрешение в файл `AndroidManifest.xml`.
3. Развернем подписанный apk в Google Developers Console.

4. Определим свои товары.
5. Реализуем код.
6. Протестируем осуществление встроенных покупок (опционально).

Шаг 1:

Прежде всего необходимо добавить файл AIDL в проект, как это описано в документации Google: https://developer.android.com/google/play/billing/billing_integrate.html#billing-add-aidl.

`IInAppBillingService.aidl` – это файл Android Interface Definition Language (AIDL), определяющий интерфейс сервиса встроенных покупок In-app Billing Version 3. Этот интерфейс используется для выполнения запросов на выставление счетов путем вызова IPC-методов.

Шаг 2:

После добавления AIDL-файла добавьте в `AndroidManifest.xml` разрешение на выставление счетов BILLING:

```
<!-- Необходимое разрешение для реализации In-app Billing -->
<uses-permission android:name="com.android.vending.BILLING" />
```

Шаг 3:

Сгенерируйте подписанный apk и загрузите его в Google Developers Console. Это необходимо для того, чтобы мы могли начать определять в нем наши товары в приложении.

Шаг 4:

Определите все ваши товары с различными `productID` и установите для каждого из них цену. Существует два типа товаров: управляемые товары (Managed Products) и подписки (Subscriptions). Как мы уже говорили, мы собираемся реализовать 4 различных расходуемых управляемых товара `item1`, `item2`, `item3`, `item4`.

Шаг 5:

После выполнения всех описанных выше действий вы готовы приступить к реализации самого кода в своей активности.

MainActivity:

```
public class MainActivity extends Activity {

    IInAppBillingService inAppBillingService;
    ServiceConnection serviceConnection;

    // productID для каждого товара. Их необходимо определить в Google Developers
    // Console.
    final String item1 = "item1";
    final String item2 = "item2";
    final String item3 = "item3";
    final String item4 = "item4";

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

        // Инстанцировать представления в соответствии с файлом макета.
        final Button buy1 = (Button) findViewById(R.id.buy1);
        final Button buy2 = (Button) findViewById(R.id.buy2);
        final Button buy3 = (Button) findViewById(R.id.buy3);
        final Button buy4 = (Button) findViewById(R.id.buy4);
```

```
// устанавливаем для каждой кнопки setOnItemClickListener().  
// buyItem() - метод, который мы будем реализовывать для запуска потока  
покупок PurchaseFlow.  
buy1.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        buyItem(item1);  
    }  
});  
  
buy2.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        buyItem(item2);  
    }  
});  
  
buy3.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        buyItem(item3);  
    }  
});  
  
buy4.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        buyItem(item4);  
    }  
});  
  
// Прикрепляем соединение ServiceConnection.  
serviceConnection = new ServiceConnection() {  
    @Override  
    public void onServiceDisconnected(ComponentName name) {  
        inAppBillingService = null;  
    }  
  
    @Override  
    public void onServiceConnected(ComponentName name, IBinder service) {  
        inAppBillingService = IInAppBillingService.Stub.  
asInterface(service);  
    }  
};  
  
// Привязываем службу.  
Intent serviceIntent = new Intent("com.android.vending.billing.  
InAppBillingService.BIND");  
serviceIntent.setPackage("com.android.vending");  
bindService(serviceIntent, serviceConnection, BIND_AUTO_CREATE);  
  
// Получение цены каждого товара и установка цены в виде текста  
// в каждой кнопке, чтобы пользователь знал цену каждого товара.  
if (inAppBillingService != null) {  
    // Внимание: необходимо создать здесь новый поток, потому что  
    // getSkuDetails() запускает сетевой запрос, который может  
    // вызвать задержку в работе приложения, если осуществлять вызов  
    // из главного потока.  
    Thread thread = new Thread(new Runnable() {
```

```

    @Override
    public void run() {
        ArrayList<String> skuList = new ArrayList<>();
        skuList.add(item1);
        skuList.add(item2);
        skuList.add(item3);
        skuList.add(item4);
        Bundle querySkus = new Bundle();
        querySkus.putStringArrayList("ITEM_ID_LIST", skuList);

        try {
            Bundle skuDetails = inAppBillingService.getSkuDetails(3,
getPackageName(), "inapp", querySkus);
            int response = skuDetails.getInt("RESPONSE_CODE");

            if (response == 0) {
                ArrayList<String> responseList = skuDetails.
getStringArrayList("DETAILS_LIST");
                for (String thisResponse : responseList) {
                    JSONObject object = new JSONObject(thisResponse);
                    String sku = object.getString("productId");
                    String price = object.getString("price");
                    switch (sku) {
                        case item1:
                            buy1.setText(price);
                            break;
                        case item2:
                            buy2.setText(price);
                            break;
                        case item3:
                            buy3.setText(price);
                            break;
                        case item4:
                            buy4.setText(price);
                            break;
                    }
                }
            }
        } catch (RemoteException | JSONException e) {
            e.printStackTrace();
        }
    }
}

```

После внедрения кода вы можете приступить к предоставлению пользователю товаров в режиме тестирования. Для этого необходимо опубликовать приложение в Google Play и активировать его.

Более подробную информацию о том, как опубликовать приложение на Google Play, можно найти по адресу: <https://developer.android.com/google/play/publishing/appdistribution/testflight.html>.

99.2. Использование встроенных покупок

```

    // Запускаем поток PurchaseFlow, передавая в качестве параметра productID
    // товара, который пользователь хочет купить.
    private void buyItem(String productID) {
        if (inAppBillingService != null) {
            try {
                Bundle buyIntentBundle = inAppBillingService.getBuyIntent(3,
getPackageName(), productID, "inapp", "bGoa+V7g/yqDXvKRqq+JTFn4uQZbPiQJo4pf9RzJ");
                PendingIntent pendingIntent = buyIntentBundle.getParcelable("BUY_
INTENT");
                startIntentSenderForResult(pendingIntent.getIntentSender(), 1003,
new Intent(), 0, 0, 0);
            }
        }
    }
}

```

```
        } catch (RemoteException | IntentSender.SendIntentException e) {
            e.printStackTrace();
        }
    }

// Отвязываем службу в функции onDestroy(). Если не отвязать, то открытое
// соединение службы может привести к снижению производительности устройства.

@Override
public void onDestroy() {
    super.onDestroy();
    if (inAppBillingService != null) {
        unbindService(serviceConnection);
    }
}

// Здесь проверяется, была ли покупка в приложении успешной. Если покупка
// была успешной, то продукт потребляется, что позволяет приложению
// вносить необходимые изменения.

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == 1003 && resultCode == RESULT_OK) {

        final String purchaseData = data.getStringExtra("INAPP_PURCHASE_DATA");

        // Внимание: тут нужно создать новый поток, потому что
        // consumePurchase() запускает сетевой запрос, что может
        // вызвать задержку в работе приложения, если осуществлять вызов
        // из главного потока.
        Thread thread = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    JSONObject jo = new JSONObject(purchaseData);
                    // Get the productID of the purchased item.
                    String sku = jo.getString("productId");
                    String productName = null;

                    // increaseCoins() - это метод, используемый в качестве
                    // примера в игре для увеличения внутриигровой валюты,
                    // если покупка была успешной.
                    // Здесь следует реализовать свой собственный код,
                    // а приложение пусть применяет
                    // необходимые изменения после успешной покупки.
                    switch (sku) {
                        case item1:
                            productName = "Item 1";
                            increaseCoins(2000);
                            break;
                        case item2:
                            productName = "Item 2";
                            increaseCoins(8000);
                            break;
                        case item3:
                            productName = "Item 3";
                            increaseCoins(18000);
                    }
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }
        });
        thread.start();
    }
}
```

```
        break;
    case item4:
        productName = "Item 4";
        increaseCoins(30000);
        break;
    }

    // Потребление покупки, чтобы пользователь мог вновь
    приобрести тот же самый товар.
    inAppBillingService.consumePurchase(3, getPackageName(),
    jo.getString("purchaseToken"));
    Toast.makeText(MainActivity.this, productName + " успешно
куплен. Прекрасный выбор!", Toast.LENGTH_LONG).show();
} catch (JSONException | RemoteException e) {
    Toast.makeText(MainActivity.this, "Не удалось обработать
данные покупки.", Toast.LENGTH_LONG).show();
    e.printStackTrace();
}
}
});  
thread.start();
}
}
```

Шаг 6:

После внедрения кода вы можете протестировать его, разместив свой apk на канале beta/alpha и предоставив возможность другим пользователям протестировать код за вас. Однако в режиме тестирования нельзя совершать реальные покупки в приложении. Сначала необходимо опубликовать приложение или игру в Play Store, чтобы все продукты были полностью активированы.

Более подробную информацию о тестировании встроенных покупок можно найти по адресу: https://developer.android.com/google/play/billing/billing_testing.html.

99.2. Использование сторонней библиотеки In-App v3

Шаг 1: Прежде всего выполните следующие два действия для добавления функциональности приложения.

1. Добавьте библиотеку с помощью:

```
repositories {  
    mavenCentral()  
}  
dependencies {  
    compile 'com.anjlab.android.iab.v3:library:1.0.+'  
}
```

2. Добавьте разрешение в файл манифеста:

```
<uses-permission android:name="com.android.vending.BILLING" />
```

Шаг 2: Инициализация обработчика покупок:

```
BillingProcessor bp = new BillingProcessor(this, "Здесь ваш лицензионный ключ из GOOGLE PLAY CONSOLE", this);
```

Для реализации обработчика покупок используем `BillingProcessor.IBillingHandler`, который содержит 4 метода:

- `onBillingInitialized()`;
- `onProductPurchased(String productId, TransactionDetails details)`: здесь необходимо обработать действия, которые должны быть выполнены после успешной покупки
- `onBillingError(int errorCode, Throwable error)`: обработка ошибок, возникших в процессе покупки
- `onPurchaseHistoryRestored()`: для восстановления покупок в приложении

Шаг 3: Как приобрести товар:

1) приобретение управляемого товара:

```
bp.purchase(YOUR_ACTIVITY, "Здесь ваш PRODUCT ID из GOOGLE PLAY CONSOLE");
```

2) приобретение подписки:

```
bp.subscribe(YOUR_ACTIVITY, "Здесь ваш SUBSCRIPTION ID из GOOGLE PLAY CONSOLE");
```

Шаг 4: Потребление товара:

Для потребления достаточно вызвать метод `consumePurchase`. `bp.consumePurchase("Здесь Ваш PRODUCT ID из GOOGLE PLAY CONSOLE");`

На Github можно также найти другие методы, связанные со встроенными покупками:
<https://github.com/anjlab/android-inapp-billing-v3/releases>.

Глава 100. Плавающая кнопка действия

Параметр

`android.support.design:elevation`

Подробности

Значение возвышения (elevation) для кнопки. Может быть ссылкой на другой ресурс в виде "@[+][package:]type/name" или атрибутом темы в виде "?[package:]type/name"

`android.support.design:fabSize`

Размер кнопки

`android.support.design:rippleColor`

Цвет ряби (ripple) для кнопки

`android.support.design:useCompatPadding`

Включение совместимых подкладок (padding)

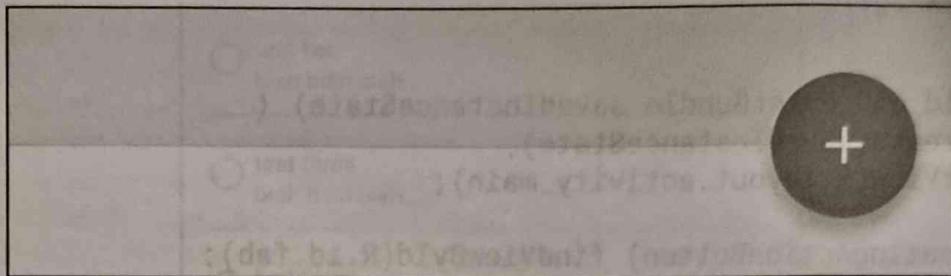
Плавающая кнопка действия (Floating Action Button, FAB) используется для особого типа действий, по умолчанию она анимируется на экране в виде расширяющегося элемента материала. Иконка внутри нее может быть анимирована, также FAB может перемещаться иначе, чем другие элементы пользовательского интерфейса, из-за их относительной важности. Плавающая кнопка действия представляет собой основное действие в приложении, которое может просто вызывать действие или осуществлять навигацию.

100.1. Как добавить плавающую кнопку действия в макет

Для использования плавающей кнопки действия достаточно добавить зависимость в файл `build.gradle`. Затем добавьте в макет:

```
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|end"
    android:layout_margin="@dimen/fab_margin"
    android:src="@drawable/my_icon" />
```

Пример:



Цвет

Цвет фона этого представления по умолчанию соответствует цвету `colorAccent` вашей темы.

На приведенном выше изображении, если `src` указывает только на значок + (по умолчанию 24x24 dp), для получения цвета фона круга можно использовать `app:backgroundTint="@color/your_colour"`.

Если необходимо изменить цвет в коде, можно использовать

```
myFab.setBackgroundTintList(ColorStateList.valueOf(ваш цвет в int));
```

Если вы хотите изменить цвет FAB в нажатом состоянии, используйте

```
mFab.setRippleColor(ваш цвет в int);
```

Позиционирование

На мобильных устройствах рекомендуется использовать не менее 16 dp отступа от края, на планшетах и настольных компьютерах – не менее 24 dp отступа.

Примечание: после того как вы установили `src`-изображение, которое должно занимать всю площадь `FloatingActionButton`, убедитесь, что вы выбрали правильный размер этого изображения, чтобы получить наилучший результат.

Размер круга по умолчанию – 56 × 56 dp.



Размер мини-круга: 40 × 40 dp.

Если вы хотите изменить только внутренний значок, используйте значок размером 24 x 24 dp по умолчанию.

100.2. Показ и скрытие плавающей кнопки действия при пролистывании

Чтобы показать и скрыть кнопку `FloatingActionButton` с анимацией по умолчанию, достаточно вызвать методы `show()` и `hide()`. Рекомендуется держать такую кнопку в макете активности, а не помещать ее во фрагмент, это позволяет использовать анимации по умолчанию при показе и скрытии.

Приведем пример с использованием ViewPager:

- Используем три вкладки (Tabs)
- Покажем кнопку FloatingActionButton для первой и третьей вкладок
- Скроем кнопку FloatingActionButton на средней вкладке

```
public class MainActivity extends AppCompatActivity {

    FloatingActionButton fab;
    ViewPager viewPager;

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

        fab = (FloatingActionButton) findViewById(R.id.fab);
        viewPager = (ViewPager) findViewById(R.id.viewpager);

        // ..... настройка ViewPager .....

        viewPager.addOnPageChangeListener(new ViewPager.OnPageChangeListener() {

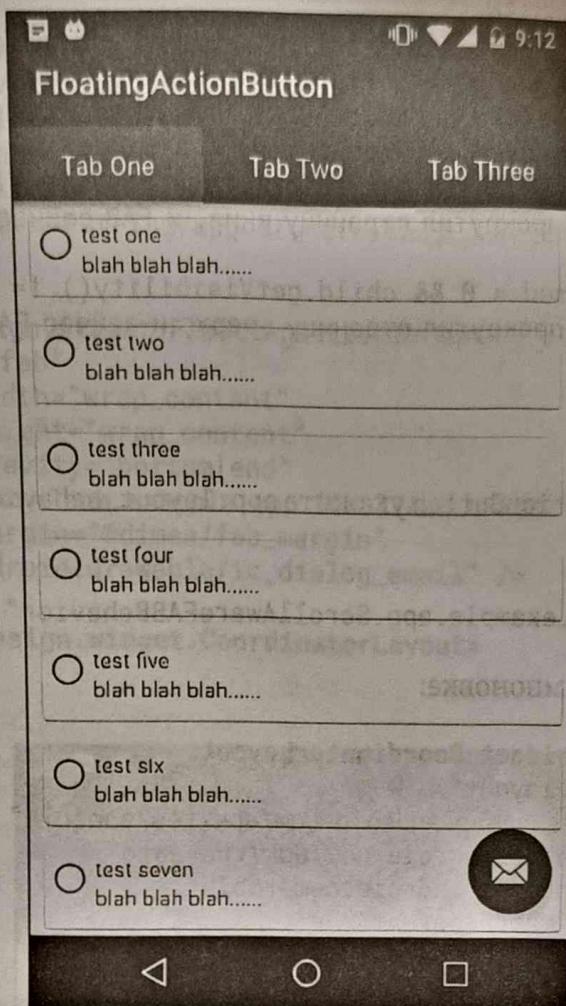
            @Override
            public void onPageSelected(int position) {
                if (position == 0) {
                    fab.setImageResource(android.R.drawable.ic_dialog_email);
                    fab.show();
                } else if (position == 2) {
                    fab.setImageResource(android.R.drawable.ic_dialog_map);
                    fab.show();
                } else {
                    fab.hide();
                }
            }

            @Override
            public void onPageScrolled(int position, float positionOffset, int
positionOffsetPixels) {}

            @Override
            public void onPageScrollStateChanged(int state) {}
        });

        // Обработка события нажатия кнопки FloatingActionButton:
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                int position = viewPager.getCurrentItem();
                if (position == 0) {
                    openSend();
                } else if (position == 2) {
                    openMap();
                }
            }
        });
    }
}
```

Результат:



100.3. Показ и скрытие плавающей кнопки действия при прокрутке

Существует возможность показывать и скрывать кнопку FloatingActionButton из поведения прокрутки с помощью подкласса FloatingActionButton.Behavior, использующего методы show() и hide(). Обратите внимание, что это работает только с CoordinatorLayout в сочетании с внутренними представлениями, поддерживающимиложенную прокрутку, такими как RecyclerView и NestedScrollView. Нижеприведенный класс ScrollAwareFABBehavior взят из руководства по Android на Codepath: https://github.com/codepath/android_guides/wiki/Floating-Action-Buttons.

```
public class ScrollAwareFABBehavior extends FloatingActionButton.Behavior {
    public ScrollAwareFABBehavior(Context context, AttributeSet attrs) {
        super();
    }

    @Override
    public boolean onStartNestedScroll(Final CoordinatorLayout coordinatorLayout,
        final FloatingActionButton child, final View directTargetChild, final View target, final int nestedScrollAxes) {
        // Обеспечение реакции на вертикальную прокрутку
        return nestedScrollAxes == ViewCompat.SCROLL_AXIS_VERTICAL
            || super.onStartNestedScroll(coordinatorLayout, child,
                directTargetChild, target, nestedScrollAxes);
    }

    @Override
```

```
public void onNestedScroll(CoordinatorLayout coordinatorLayout, final
    FloatingActionButton child,
        final View target, final int dxConsumed, final int dyConsumed,
        final int dxUnconsumed, final int dyUnconsumed) {
    super.onNestedScroll(coordinatorLayout, child, target, dxConsumed,
        dyConsumed, dxUnconsumed, dyUnconsumed);
    if (dyConsumed > 0 && child.getVisibility() == View.VISIBLE) {
        // Пользователь прокрутил страницу вниз, и FAB сейчас видна -> скрыть FAB
        child.hide();
    } else if (dyConsumed < 0 && child.getVisibility() != View.VISIBLE) {
        // Пользователь прокрутил страницу вверх, и сейчас FAB не видна -> показать
        child.show();
    }
}
```

В xml макета FloatingActionButton укажите app:layout_behavior с полным именем класса ScrollAwareFABBehavior:

```
app:layout_behavior="com.example.app.ScrollAwareFABBehavior"
```

Например, при такой компоновке:

```
<android.support.design.widget.CoordinatorLayout  
    android:id="@+id/main_layout"  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context=".MainActivity">
```

```
<android.support.design.widget.AppBarLayout
```

```
        android:id="@+id/appBarLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" app:elevation="6dp">
<android.support.v7.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:background="?attr/colorPrimary"
    android:minHeight="?attr/actionBarSize"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light"
    app:elevation="0dp"
    app:layout_scrollFlags="scroll|enterAlways"
/>
```

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

```
        android:id="@+id/tab_layout"
        app:tabMode="fixed"
        android:layout_below="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="?attr/colorPrimary"
        app:elevation="0dp"
        app:tabTextColor="#d3d3d3"
        android:minHeight="?attr/actionBarSize"
    />
```

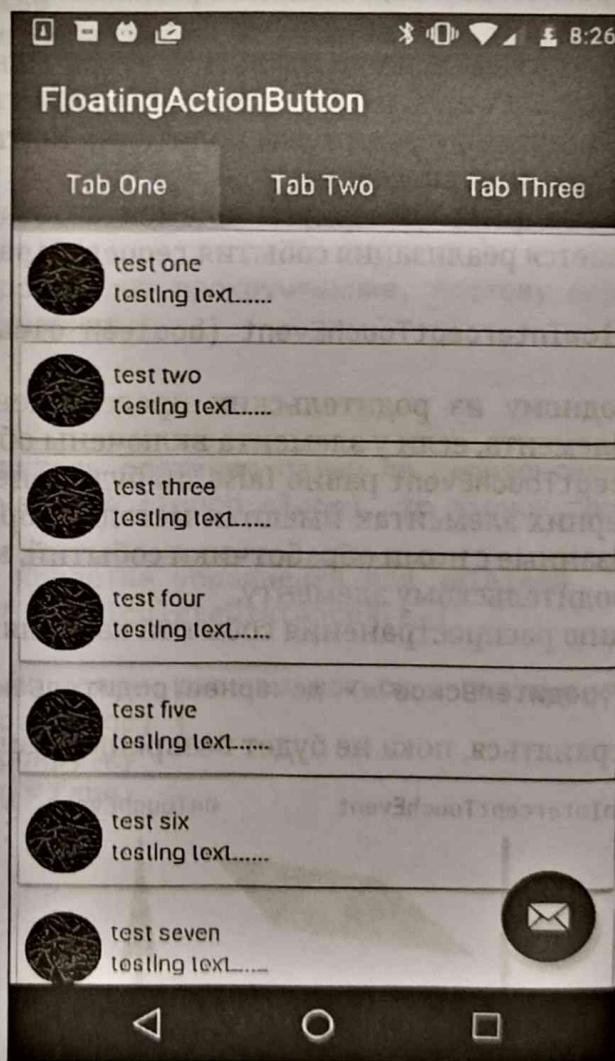
```
</android.support.design.widget.AppBarLayout>

<android.support.v4.view.ViewPager
    android:id="@+id/viewpager"
    android:layout_below="@+id/tab_layout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    />

<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|end"
    app:layout_behavior="com.example.app.ScrollAwareFABBehavior"
    android:layout_margin="@dimen/fab_margin"
    android:src="@android:drawable/ic_dialog_email" />

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

Вот результат:



100.4. Настройка поведения кнопки FAB

Вы можете задать поведение FAB в XML-файле. Например:

```
<android.support.design.widget.FloatingActionButton
    app:layout_behavior=".MyBehavior" />
```

...или программно с помощью следующего кода:

```
CoordinatorLayout.LayoutParams p = (CoordinatorLayout.LayoutParams) fab.  
getLayoutParams();  
p.setBehavior(xxxx);  
fab.setLayoutParams(p);
```

Глава 101. События касания

101.1. Как различать события касания дочерней и родительской групп представлений

1. Управление событиями `onTouchEvent()` для вложенных групп представлений может осуществляться с помощью метода `onInterceptTouchEvent`, возвращающего булево значение.

Значение по умолчанию для `OnInterceptTouchEvent` равно `false`. Родительское событие `onTouchEvent` принимается раньше дочернего. Если `OnInterceptTouchEvent` возвращает `false`, то событие движения отправляется вниз по цепочке в обработчик `OnTouchEvent` дочернего объекта. Если возвращается `true`, то именно родительский обработчик будет обрабатывать это событие касания.

Однако могут возникнуть ситуации, когда мы хотим, чтобы некоторые дочерние элементы управляли событиями `OnTouchEvent`, и при этом другие события управлялись родительским представлением (или, возможно, родителем родителя). Контроль над этим процессом может осуществляться несколькими способами.

2. Одним из способов защиты дочернего элемента от родительского события `OnInterceptTouchEvent` является реализация события `requestDisallowInterceptTouchEvent`.

```
public void requestDisallowInterceptTouchEvent (boolean disallowIntercept)
```

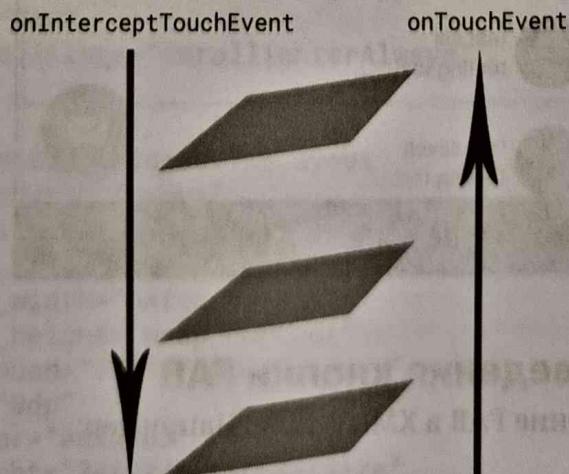
Это не позволяет ни одному из родительских представлений управлять событием `OnTouchEvent` для данного элемента, если у элемента включены обработчики событий.

Если значение `OnInterceptTouchEvent` равно `false`, то будет оценено `OnTouchEvent` дочернего элемента. Если в дочерних элементах имеются методы, обрабатывающие различные события касания, то все связанные с ними обработчики событий, которые отключены, будут возвращать `OnTouchEvent` родительскому элементу.

Рассмотрим визуализацию распространения событий касания:

родительское -> дочернее | родительское -> дочернее | родительское -> дочернее

События будут распространяться, пока не будет возвращено значение `true`.



3. Другой способ – возврат изменяющихся значений из события OnInterceptTouchEvent для родителя.

Этот пример с ресурса <http://developer.android.com/training/gestures/viewgroup.html> показывает, как перехватить дочернее событие OnTouchEvent, когда пользователь выполняет прокрутку.

a)

```
@Override
public boolean onInterceptTouchEvent(MotionEvent ev) {
    /*
     * Этот метод определяет, хотим ли мы перехватить движение.
     * Если возвращается true, то будет вызван onTouchEvent, и мы выполним там
     * фактическое действие прокрутки.
     */
    final int action = MotionEventCompat.getActionMasked(ev);

    // Всегда обрабатывайте случай, когда жест касания завершен.
    if (action == MotionEvent.ACTION_CANCEL || action == MotionEvent.ACTION_UP) {
        // Отпустить прокрутку.
        mIsScrolling = false;
        return false; // Не перехватываем событие касания, пусть им займется дочернее
                     // представление
    }

    switch (action) {
        case MotionEvent.ACTION_MOVE: {
            if (mIsScrolling) {
                // Сейчас происходит прокручивание, поэтому перехватываем
                // событие касания
                return true;
            }

            // Если пользователь протащил палец по горизонтали больше,
            // чем наклон касания (touch slope), то запуск прокрутки
            // оставлено в качестве упражнения для читателя
            final int xDiff = calculateDistanceX(ev);

            // Наклон касания должен рассчитываться с помощью констант ViewConfiguration
            if (xDiff > mTouchSlop) {
                // Начать прокрутку
                mIsScrolling = true;
                return true;
            }
        } break;
    }
}

// В общем случае мы не хотим перехватывать события касания. Они должны быть
// обработаны дочерним представлением.
return false;
}
```

Далее приведен код из того же источника, показывающий, как создать параметры прямоугольника вокруг вашего элемента:

б)

```

// Прямоугольник для кнопки ImageButton
myButton.getHitRect(delegateArea);

// Расширение области касания кнопки ImageButton за ее пределы
// вправо и вниз.
delegateArea.right += 100;
delegateArea.bottom += 100;

// Инстанцируем делегат TouchDelegate.
// "delegateArea" - это границы в локальных координатах содержащего
// представления, которые должны быть сопоставлены с представлением делегата.
// "myButton" - это дочернее представление, которое должно получать
// события движения.
TouchDelegate touchDelegate = new TouchDelegate(delegateArea, myButton);

// Устанавливаем TouchDelegate на родительском представлении так, чтобы касания
// в пределах границ делегата касания направлялись к дочернему объекту.
if (View.class.isInstance(myButton.getParent())) {
    ((View) myButton.getParent()).setTouchDelegate(touchDelegate);
}

```

Глава 102. Обработка событий касания и движения

Слушатель

onTouchListener

Подробности

Обрабатывает одиночные касания для кнопок, поверхностей и т. д.

onTouchEvent

Слушатель, который используется в поверхностях (например SurfaceView). Не требует установки, как другие слушатели (например onTouchListener).

onLongTouch

Аналогичен onTouch, но прослушивает длительные нажатия на кнопки, поверхности и т. д.

Эта глава содержит краткое описание некоторых основных систем обработки касаний и движений в API Android.

102.1. Кнопки

События касания, связанные с кнопкой (Button), могут быть проверены следующим образом:

```

public class ExampleClass extends Activity implements View.OnClickListener, View.
OnLongClickListener {
    public Button onLong, onClick;

    @Override
    public void onCreate(Bundle sis) {
        super.onCreate(sis);
        setContentView(R.layout.layout);
        onLong = (Button) findViewById(R.id.onLong);
        onClick = (Button) findViewById(R.id.onClick);
    }
}

```

```
// Кнопки созданы. Теперь нам нужно сообщить системе,
// что у этих кнопок есть слушатель для проверки событий касания.
// "this" относится к данному классу, поскольку он содержит соответствующие
// слушатели событий.
onLong.setOnLongClickListener(this);
onClick.setOnClickListener(this);
```

[OR]

```
onClick.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View v){
        // Этот слушатель предназначен только для одной кнопки.
        // Это означает, что никакие другие входные данные сюда не попадут.
        // Это делает оператор switch ненужным здесь.
    }
});
```

```
onLong.setOnLongClickListener(new View.OnLongClickListener(){
    @Override
    public boolean onLongClick(View v){
        // См. комментарий к onClick.setOnClickListener().
    }
});
```

```
@Override
public void onClick(View v) {
    // Если необходимо обработать несколько кнопок, используйте switch для их
    // обработки.
    switch(v.getId()){
        case R.id.onClick:
            // Принять меры.
            break;
    }
}
```

```
@Override
public boolean onLongClick(View v) {
    // Если необходимо обработать несколько кнопок, используйте switch для их
    // обработки.
    switch(v.getId()){
        case R.id.onLong:
            // Принять меры.
            break;
    }
}
```

102.2. Поверхности

Обработчик событий касания для поверхностей (например, SurfaceView, GLSurfaceView и других):

```
import android.app.Activity;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.SurfaceView;
```

```

import android.view.View;

public class ExampleClass extends Activity implements View.OnTouchListener{
    @Override
    public void onCreate(Bundle sis){
        super.onCreate(sis);
        CustomSurfaceView csv = new CustomSurfaceView(this);
        csv.setOnTouchListener(this);
        setContentView(csv);
    }

    @Override
    public boolean onTouch(View v, MotionEvent event) {
        // Добавьте switch (см. пример с кнопками), если вы работаете с несколькими
        // представлениями.
        // Здесь можно увидеть (используя событие MotionEvent), что событие касания
        // происходит, и узнать, что это за касание.
        return false;
    }
}

```

Альтернативный вариант (на поверхности):

```

public class CustomSurfaceView extends SurfaceView {
    @Override
    public boolean onTouchEvent(MotionEvent ev) {
        super.onTouchEvent(ev);
        // Здесь обрабатываются события касания. При этом не нужно вызывать
        // слушателя. Обратите внимание, что этот слушатель применяется только к той
        // поверхности, на которой он размещен (в данном случае CustomSurfaceView),
        // что означает, что все остальное, на что нажимается вне SurfaceView,
        // обрабатывается теми частями вашего приложения, у которых есть в этой
        // области слушатель.
        return true;
    }
}

```

102.3. Обработка множественного касания поверхности

```

public class CustomSurfaceView extends SurfaceView {
    @Override
    public boolean onTouchEvent(MotionEvent e) {
        super.onTouchEvent(e);
        if(e.getPointerCount() > 2){
            return false;// Если мы хотим ограничить количество указателей (pointers),
            // то возвращаем false, что запрещает использование указателя. На него
            // также не будет реакции, связанной с любыми событиями касания,
            // пока не произойдет снятие и новое нажатие.
        }

        // Здесь можно проверить, является ли количество указателей равным [x],
        // и предпринять соответствующие действия, если указатель убран, используется
        // новый или [x] указателей перемещается.

        // Некоторые примеры обработки и т.п. событий касания/движения.

        switch (MotionEventCompat.getActionMasked(e)) {
            case MotionEvent.ACTION_DOWN:
            case MotionEvent.ACTION_POINTER_DOWN:
                // Один или несколько указателей касаются экрана.
                break;
        }
    }
}

```

```

        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_POINTER_UP:
            // Один или несколько указателей перестают касаться экрана.
            break;
        case MotionEvent.ACTION_MOVE:
            // Перемещение одного или нескольких указателей.
            if(e.getPointerCount() == 2){
                move();
            }else if(e.getPointerCount() == 1){
                paint();
            }else{
                zoom();
            }
            break;
    }
    return true; // Разрешить повторное действие.
}
}

```

Глава 103. Обнаружение событий встремивания в Android

103.1. Детектор тряски

```

public class ShakeDetector implements SensorEventListener {
    private static final float SHAKE_THRESHOLD_GRAVITY = 2.7F;
    private static final int SHAKE_SLOP_TIME_MS = 500;
    private static final int SHAKE_COUNT_RESET_TIME_MS = 3000;

    private OnShakeListener mListener; private long mShakeTimestamp; private int
    mShakeCount;
    public void setOnShakeListener(OnShakeListener listener) {
        this.mListener = listener;
    }

    public interface OnShakeListener {
        public void onShake(int count);
    }

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

    @Override
    public void onSensorChanged(SensorEvent event) {
        if (mListener != null) {
            float x = event.values[0];
        }
    }
}

```

```
    float y = event.values[1];
    float z = event.values[2];

    float gX = x / SensorManager.GRAVITY_EARTH;
    float gY = y / SensorManager.GRAVITY_EARTH;
    float gZ = z / SensorManager.GRAVITY_EARTH;

    // При отсутствии движения gForce будет близка к 1.
    float gForce = FloatMath.sqrt(gX * gX + gY * gY + gZ * gZ);

    if (gForce > SHAKE_THRESHOLD_GRAVITY) {
        final long now = System.currentTimeMillis();
        // игнорировать события встряхивания, происходящие слишком близко друг
        // к другу (500 мс)
        if (mShakeTimestamp + SHAKE_SLOP_TIME_MS > now) {
            return;
        }

        // сбросить счетчик встряхиваний после 3 секунд отсутствия встряхиваний
        if (mShakeTimestamp + SHAKE_COUNT_RESET_TIME_MS < now) {
            mShakeCount = 0;
        }

        mShakeTimestamp = now;
        mShakeCount++;

        mListener.onShake(mShakeCount);
    }
}
```



103.2. Использование системы обнаружения встряхиваний Seismic

Seismic – это библиотека обнаружения встряхиваний Android-устройств от компании Square (<https://github.com/square/seismic>). Чтобы использовать ее, просто начните прослушивать события от встряхиваний, которые она испускает.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    sm = (SensorManager) getSystemService(SENSOR_SERVICE);
    sd = new ShakeDetector(() -> { /* реакция на обнаружено встряхивание */ });
}

@Override
protected void onResume() {
    sd.start(sm);
}

@Override
protected void onPause() {
    sd.stop();
}
```

Для задания другого порога ускорения используйте `sd.setSensitivity(sensitivity)` с чувствительностью `SENSITIVITY_LIGHT`, `SENSITIVITY_MEDIUM`, `SENSITIVITY_HARD` или любое

другое разумное целочисленное значение. Приведенные значения по умолчанию лежат в диапазоне от 11 до 15.

Установка

```
compile 'com.squareup:seismic:1.0.2'
```

```
public void handleEvent(BarcodeEvent event) {
    base.setBackground(result, R.drawable.tiny_barcode,
        LENGTH_SHORT).show();
}
```

Глава 104. GreenRobot EventBus

Режим потока

`ThreadMode.POSTING`

Описание

Будет вызван в том же потоке, в котором было опубликовано событие. Это режим по умолчанию.

`ThreadMode.MAIN`

Будет вызываться в основном потоке пользовательского интерфейса.

`ThreadMode.BACKGROUND`

Будет вызываться в фоновом потоке. Если поток сообщений не является основным, то он будет использоваться.

При размещении на основном потоке EventBus имеет единственный фоновый поток, который он будет использовать.

`ThreadMode.ASYNC`

Будет вызываться в собственном потоке.

104.1. Передача простого события

Первое, что нам нужно сделать, это добавить EventBus в gradle-файл нашего модуля:

```
dependencies {
    ...
    compile 'org.greenrobot:eventbus:3.0.0'
    ...
}
```

Теперь нам необходимо создать модель для нашего события. Она может содержать все, что мы хотим передать. Пока мы просто создадим пустой класс:

```
public class DeviceConnectedEvent
```

Теперь мы можем добавить в нашу Activity код, который будет регистрироваться в EventBus и подписываться на событие:

```
public class MainActivity extends AppCompatActivity
{
    private EventBus _eventBus;

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

        _eventBus = EventBus.getDefault();
    }
}
```

```

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

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

@Subscribe(threadMode = ThreadMode.MAIN)
public void onDeviceConnected (final DeviceConnectedEvent event)
{
    // Обработка события и обновление пользовательского интерфейса
}
}

```

В этой активности мы получаем экземпляр EventBus в методе onCreate(). Мы регистрируемся и снимаем регистрацию для событий в onStart() и onStop() соответственно. Важно помнить о необходимости снятия регистрации, когда слушатель теряет область видимости, иначе возможна утечка информации из активности.

Теперь мы определяем метод, который мы хотим вызывать вместе с событием. Аннотация @Subscribe указывает шине событий EventBus, какие методы можно искать для обработки событий. Для регистрации в EventBus необходимо иметь хотя бы один метод, аннотированный @Subscribe, иначе будет выброшено исключение. В аннотации мы определяем режим потока. Это указывает EventBus, в каком потоке вызывать метод. Это очень удобный способ передачи информации из фонового потока в поток пользовательского интерфейса. Именно это мы и делаем здесь. ThreadMode.MAIN означает, что этот метод будет вызван в основном потоке пользовательского интерфейса Android, поэтому здесь можно смело выполнять любые манипуляции с пользовательским интерфейсом. Название метода не имеет значения. Единственное, что (кроме аннотации @Subscribe) интересует EventBus, – это тип аргумента. Если тип совпадает, то метод будет вызван при получении события.

И последнее, что нам нужно сделать, это опубликовать событие. Этот код будет находиться в нашей службе Service.

```
EventBus.getDefault().post(new DeviceConnectedEvent());
```

Теперь EventBus возьмет это событие DeviceConnectedEvent, просмотрит своих зарегистрированных слушателей, просмотрит методы, на которые они подписались, найдет те, которые принимают DeviceConnectedEvent в качестве аргумента, и вызовет их в том потоке, в котором они должны быть вызваны.

104.2. События приема

Для получения событий необходимо зарегистрировать свой класс на шине событий EventBus.

```

@Override
public void onStart() {
    super.onStart();
    EventBus.getDefault().register(this);
}

@Override
public void onStop() {

```

```
EventBus.getDefault().unregister(this);
super.onStop();
}
```

...а затем подписаться на события:

```
@Subscribe(threadMode = ThreadMode.MAIN)
public void handleEvent(ArbitraryEvent event) {
    Toast.makeText(getApplicationContext(), "Тип события: " + event.getEventType(), Toast.LENGTH_SHORT).show();
}
```

104.3. Отправка событий

Отправлять события так же просто, как создавать объект Event и затем отправлять его.

```
EventBus.getDefault().post(new ArbitraryEvent(ArbitraryEvent.TYPE_1));
```

Глава 105. Вибрация

105.1. Начало работы с вибрацией

Получение разрешения на вибрацию

Прежде чем приступить к реализации кода, необходимо добавить разрешение в манифест Android:

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

Импорт библиотеки вибраций

```
import android.os.Vibrator;
```

Получение экземпляра Vibrator из контекста

```
Vibrator vibrator = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
```

Проверка наличия Vibrator в устройстве

```
void boolean isHaveVibrate(){
    if (vibrator.hasVibrator()) {
        return true;
    }
    return false;
}
```

105.2. Вибрация с неопределенной продолжительностью

Используем vibrate(long[] pattern, int repeat):

```
Vibrator vibrator = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);

// Задержка начала отсчета времени
// Фаза вибрации в течение 500 миллисекунд
```



```
// Фаза сна в течение 1000 миллисекунд
long[] pattern = {0, 500, 1000};

// значение 0 - повторять бесконечно
vibrator.vibrate(pattern, 0);
```

105.3. Шаблоны вибрации

Вы можете создавать шаблоны вибрации, передавая массив чисел типа long, каждое из которых представляет собой длительность в миллисекундах. Первое число – это время задержки запуска. Затем каждый элемент массива чередуется между фазами вибрации, «сна», вибрации, «сна» и т. д.

Следующий пример демонстрирует шаблон:

- виброровать 100 миллисекунд и «спать» 1000 миллисекунд
- виброровать 200 миллисекунд и «спать» 2000 миллисекунд

```
long[] pattern = {0, 100, 1000, 200, 2000};
```

Для того чтобы вызвать повтор шаблона, необходимо передать индекс массива паттернов, с которого начинается повтор, или -1 для отключения повтора.

```
Vibrator vibrator = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
vibrator.vibrate(pattern, -1); // не повторяется
vibrator.vibrate(pattern, 0); // повторяется вечно
```

105.4. Остановка вибрации

Если вы хотите остановить вибрацию, вызовите:

```
vibrator.cancel();
```

105.5. Однократная вибрация

Используйте vibrate(long milliseconds):

```
Vibrator vibrator = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
vibrator.vibrate(500);
```

Глава 106. Провайдеры контента

106.1. Реализация базового класса провайдера контента

1. Создание класса Contract Class

Класс Contract Class определяет константы, которые помогают приложениям работать с URI контента, именами колонок, действиями намерений и другими возможностями провайдера контента. Такие классы не включаются автоматически в состав провайдера; разработчик должен определить их и затем предоставить доступ к ним другим разработчикам.

Провайдер обычно имеет только одно полномочия (authority), которые служат его внутренним именем в Android. Чтобы избежать конфликтов с другими провайдерами, используйте уникальные полномочия контента.

Поскольку эта рекомендация справедлива и для имен пакетов Android, можно определить полномочия провайдера как расширение имени пакета, содержащего провайдер. На-

пример, если имя пакета Android – com.example.appname, то провайдеру следует присвоить полномочия com.example.appname.provider.

```
public class MyContract {
    public static final String CONTENT_AUTHORITY = "com.example.myApp";
    public static final String PATH_DATATABLE = "dataTable";
    public static final String TABLE_NAME = "dataTable";
}
```

URI содержимого – это URI, идентифицирующий данные в провайдере. URI содержимого включает символическое имя всего провайдера (его полномочия) и имя, указывающее на таблицу или файл (путь). Необязательная часть идентификатора указывает на отдельную строку в таблице. Каждый метод доступа к данным провайдера контента имеет в качестве аргумента URI содержимого, который позволяет определить таблицу, строку или файл для доступа. Определите их в классе Contract Class.

```
public static final Uri BASE_CONTENT_URI = Uri.parse("content://" + CONTENT_
AUTHORITY);
public static final Uri CONTENT_URI = BASE_CONTENT_URI.buildUpon().appendPath(PATH_
DATATABLE).build();

// определить все столбцы таблицы и необходимые общие функции
```

2. Создание класса-помощника

Класс-помощник управляет созданием базы данных и управлением версиями.

```
public class DatabaseHelper extends SQLiteOpenHelper {

    // Увеличение версии при изменении структуры базы данных
    public static final int DATABASE_VERSION = 1;
    // Имя базы данных в файловой системе, которое может быть любым
    public static final String DATABASE_NAME = "weather.db";

    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        // Вызывается при первом создании базы данных.
        // Должно произойти создание таблиц и их первоначальное заполнение.
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Вызывается при необходимости обновления базы данных. Реализации
        // следует использовать этот метод для удаления таблиц, добавления таблиц
        // или выполнения любых других действий, необходимых для обновления
        // до новой версии схемы.
    }
}
```

3. Создайте класс, расширяющий класс ContentProvider

```
public class MyProvider extends ContentProvider {
    public DatabaseHelper dbHelper;
    public static final UriMatcher matcher = buildUriMatcher();
```

```
public static final int DATA_TABLE = 100;
public static final int DATA_TABLE_DATE = 101;
```

UriMatcher сопоставляет полномочия и путь с целочисленным значением. Метод `match()` возвращает уникальное целочисленное значение для URI (это может быть любое произвольное число, лишь бы оно было уникальным). Оператор `switch` выбирает между запросом всей таблицы и запросом одной записи. UriMatcher возвращает 100, если URI является URI содержимого таблицы, и 101, если URI указывает на конкретную строку в этой таблице. Вы можете использовать подстановочный знак `#` для соответствия любому числу и `*` для соответствия любой строке.

```
public static UriMatcher buildUriMatcher() {
    UriMatcher uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    uriMatcher.addURI(CONTENT_AUTHORITY, MyContract.PATH_DATATABLE, DATA_TABLE);
    uriMatcher.addURI(CONTENT_AUTHORITY, MyContract.PATH_DATATABLE + "/#", DATA_TABLE_DATE);
    return uriMatcher;
}
```

Важно: порядок вызовов `addURI()` имеет значение! UriMatcher будет искать в последовательном порядке от первого добавленного к последнему. Поскольку такие подстановочные знаки, как `#` и `*`, являются «жадными», необходимо убедиться, что вы правильно упорядочили ваши URI. Например:

```
uriMatcher.addURI(CONTENT_AUTHORITY, "/example", 1);
uriMatcher.addURI(CONTENT_AUTHORITY, "/*", 2);
```

...является правильным порядком, поскольку UriMatcher сначала ищет `/example`, прежде чем прибегнуть к `/*`. Если эти вызовы методов поменять местами и вызвать `uriMatcher.match("/example")`, то UriMatcher, встретив путь `/*`, прекратит поиск совпадений и вернет неверный результат!

Затем необходимо переопределить следующие функции:

`onCreate()`: Инициализация провайдера. Система Android вызывает этот метод сразу после создания провайдера. Обратите внимание, что провайдер не создается до тех пор, пока к нему не попытается обратиться объект `ContentResolver`.

```
@Override
public boolean onCreate() {
    dbhelper = new DatabaseHelper(getContext());
    return true;
}

getType(): возвращает MIME-тип, соответствующий URI содержимого.

@Override
public String getType(Uri uri) {
    final int match = matcher.match(uri);
    switch (match) {
        case DATA_TABLE:
            return ContentResolver.CURSOR_DIR_BASE_TYPE + "/" + MyContract.CONTENT_
                AUTHORITY + "/" + MyContract.PATH_DATATABLE;
        case DATA_TABLE_DATE:
            return ContentResolver.ANY_CURSOR_ITEM_TYPE + "/" + MyContract.CONTENT_
                AUTHORITY + "/" + MyContract.PATH_DATATABLE;
        default:
            throw new UnsupportedOperationException("Unknown Uri: " + uri);
    }
}
```

`query()`: Получение данных от вашего провайдера. С помощью аргументов можно выбрать таблицу для запроса, возвращаемые строки и столбцы, а также порядок сортировки результата. Возвращает данные в виде объекта Cursor.

```
@Override
public Cursor query(Uri uri, String[] projection, String selection, String[]
selectionArgs, String sortOrder) {
    Cursor retCursor = dbHelper.getReadableDatabase().query(
        MyContract.TABLE_NAME, projection, selection, selectionArgs, null, null,
        sortOrder);
    retCursor.setNotificationUri(getContext().getContentResolver(), uri);
    return retCursor;
}
```

Вставьте новую строку в ваш провайдер. Используйте аргументы для выбора таблицы назначения и получения значений столбцов. Верните URI содержимого для новой вставленной строки.

```
@Override
public Uri insert(Uri uri, ContentValues values)
{
    final SQLiteDatabase db = dbHelper.getWritableDatabase();
    long id = db.insert(MyContract.TABLE_NAME, null, values);
    return ContentUris.withAppendedId(MyContract.CONTENT_URI, id);
}
```

`delete()`: Удаление строк из вашего провайдера. С помощью аргументов выберите таблицу и строки для удаления. Возвращается количество удаленных строк.

```
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    int rowsDeleted = db.delete(MyContract.TABLE_NAME, selection, selectionArgs);
    getContext().getContentResolver().notifyChange(uri, null);
    return rowsDeleted;
}
```

`update()`: Обновление существующих строк в вашем провайдере. Используйте аргументы для выбора таблицы и строк для обновления, а также для получения новых значений столбцов. Возвращает количество обновленных строк.

```
@Override
public int update(Uri uri, ContentValues values, String selection, String[]
selectionArgs) {
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    int rowsUpdated = db.update(MyContract.TABLE_NAME, values, selection,
        selectionArgs);
    getContext().getContentResolver().notifyChange(uri, null);
    return rowsUpdated;
}
```

4. Обновление файла манифеста.

```
<provider
    android:authorities="com.example.myApp"
    android:name=".DatabaseProvider"/>
```

Глава 107. Dagger 2

107.1. Настройка компонентов для внедрения зависимостей приложений и активностей

Базовый AppComponent, зависящий от одного AppModule для предоставления синглтон-объектов в масштабах всего приложения.

```
@Singleton
@Component(modules = AppModule.class)
public interface AppComponent {
    void inject(App app);
    Context provideContext();
    Gson provideGson();
}
```

Модуль, используемый вместе с AppComponent, который будет предоставлять свои объекты-синглтоны, например, экземпляр Gson, для повторного использования во всем приложении.

```
@Module
public class AppModule {
    private final Application mApplication;

    public AppModule(Application application) {
        mApplication = application;
    }

    @Singleton
    @Provides
    Gson provideGson() {
        return new Gson();
    }

    @Singleton
    @Provides
    Context provideContext() {
        return mApplication;
    }
}
```

Подклассифицированное приложение для настройки Dagger и синглтон-компонента.

```
public class App extends Application {
    @Inject
    AppComponent mAppComponent;

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

        DaggerAppComponent.builder().appModule(new AppModule(this)).build()
            .inject(this);
    }
}
```

```

public AppComponent getAppComponent() {
    return mAppComponent;
}
}

```

Теперь компонент активности, зависящий от AppComponent, получает доступ к синглтонным объектам.

```

@ActivityScope
@Component(dependencies = AppComponent.class, modules = ActivityModule.class)
public interface MainActivityComponent {

    void inject(MainActivity activity);
}

```

Многократно используемый модуль ActivityModule, который будет предоставлять основные зависимости, такие как FragmentManager:

```

@Module
public class ActivityModule {

    private final AppCompatActivity mActivity;

    public ActivityModule(AppCompatActivity activity) {
        mActivity = activity;
    }

    @ActivityScope
    public AppCompatActivity provideActivity() {
        return mActivity;
    }

    @ActivityScope
    public FragmentManager provideFragmentManager(AppCompatActivity activity) {
        return activity.getSupportFragmentManager();
    }
}

```

Собрав все вместе, мы можем делать внедрения зависимостей наших активностей и быть уверенными в том, что используем один и тот же Gson во всем приложении!

```

public class MainActivity extends AppCompatActivity {

    @Inject Gson mGson;

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

        DaggerMainActivityComponent.builder()
            .appComponent(((App) getApplication()).getAppComponent())
            .activityModule(new ActivityModule(this))
            .build().inject(this);
    }
}

```

107.2. Пользовательские области действия

```
@Scope
@Documented
@Retention(RUNTIME)
public @interface ActivityScope {
}
```

Области действия (Scopes) – это просто аннотации, и при необходимости вы можете создавать свои собственные.

107.3. Использование @Subcomponent вместо @Component(dependencies={...})

```
@Singleton
@Component(modules = AppModule.class)
public interface AppComponent {
    void inject(App app);

    Context provideContext();
    Gson provideGson();

    MainActivityComponent mainActivityComponent(ActivityModule activityModule);
}

@Override
public class MainActivity extends AppCompatActivity {
    @Inject
    Gson mGson;

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

        ((App) getApplication()).getAppComponent()
            .mainActivityComponent(new ActivityModule(this)).inject(this);
    }
}
```

107.4. Создание компонента из нескольких модулей

Dagger 2 поддерживает создание компонента из нескольких модулей. Вы можете создать свой компонент таким образом:

```
@Singleton
@Component(modules = {GeneralPurposeModule.class, SpecificModule.class})
public interface MyMultipleModuleComponent {
    void inject(MyFragment myFragment);
```

```

void inject(MyService myService);
void inject(MyController myController);
void inject(MyActivity myActivity);
}

```

Тогда два ссылочных модуля GeneralPurposeModule и SpecificModule могут быть реализованы следующим образом:

GeneralPurposeModule.java

```

@Module
public class GeneralPurposeModule {
    @Provides
    @Singleton
    public Retrofit getRetrofit(PropertiesReader propertiesReader,
        RetrofitHeaderInterceptor headerInterceptor){
        // Здесь программная логика...
        return retrofit;
    }

    @Provides
    @Singleton
    public PropertiesReader getPropertiesReader(){
        return new PropertiesReader();
    }

    @Provides
    @Singleton
    public RetrofitHeaderInterceptor getRetrofitHeaderInterceptor(){
        return new RetrofitHeaderInterceptor();
    }
}

```

SpecificModule.java

```

@Singleton
@Module
public class SpecificModule {
    @Provides @Singleton
    public RetrofitController getRetrofitController(Retrofit retrofit){
        RetrofitController retrofitController = new RetrofitController();
        retrofitController.setRetrofit(retrofit);
        return retrofitController;
    }

    @Provides @Singleton
    public MyService getMyService(RetrofitController retrofitController){
        MyService myService = new MyService();
        myService.setRetrofitController(retrofitController);
        return myService;
    }
}

```

На этапе внедрения зависимостей компонент будет брать объекты из обоих модулей в зависимости от потребностей.

Такой подход очень полезен с точки зрения *модульности*. В примере имеется модуль общего назначения, используемый для инстанцирования таких компонентов, как объект

Retrofit (используется для работы с сетевым взаимодействием) и PropertiesReader (отвечает за работу с конфигурационными файлами). Есть также специальный модуль, который занимается инстанцированием конкретных контроллеров и служебных классов применительно к конкретному компоненту приложения.

107.5. Как добавить Dagger 2 в build.gradle

После выхода Gradle 2.2 использование плагина android-apt больше не применяется. Следует использовать такой способ настройки Dagger 2:

Для Gradle >= 2.2

```
dependencies {
    // команда apt поступает из плагина android-apt
    annotationProcessor 'com.google.dagger:dagger-compiler:2.8'
    compile 'com.google.dagger:dagger:2.8'
    provided 'javax.annotation:jsr250-api:1.0'
}
```

107.6. Внедрение конструктора

С помощью Dagger легко могут быть созданы классы без зависимостей:

```
public class Engine {

    @Inject // <-- Аннотируйте свой конструктор.
    public Engine() {
    }
}
```

Этот класс может быть предоставлен *любым* компонентом. Он *сам не имеет зависимостей и не входит в область*. Никакого дополнительного кода не требуется.

Зависимости объявляются в конструкторе как параметры. Dagger будет вызывать конструктор и предоставлять зависимости, если они могут быть предоставлены.

```
public class Car {
    private Engine engine;

    @Inject
    public Car(Engine engine) {
        this.engine = engine;
    }
}
```

Этот класс может быть предоставлен *любым* компонентом, если этот компонент также может предоставить все свои зависимости – в данном случае Engine. Поскольку Engine также может быть внедрен конструктором, любой компонент может предоставить класс Car.

Внедрение конструктора можно использовать в тех случаях, когда все зависимости могут быть предоставлены компонентом. Компонент может предоставить зависимость, если:

- он может создать его с помощью внедрения конструктора
- модуль компонента может предоставить его
- он может быть предоставлен родительским компонентом (если это @Subcomponent)
- он может использовать объект, открытый компонентом, от которого он зависит (зависимости компонентов)

Глава 108. Realm

Мобильная база данных Realm (Realm Mobile Database, см. <https://realm.io/products/realm-mobile-database/>) – это альтернатива SQLite. Realm работает гораздо быстрее, чем ORM, и зачастую быстрее, чем SQLite.

Преимущества

Оффлайн-функциональность, высокая скорость запросов, потокобезопасность, кроссплатформенные приложения, шифрование, реактивная архитектура.

108.1. Сортированные запросы

Для сортировки запроса вместо функции `findAll()` следует использовать функцию `findAllSorted()`.

```
RealmResults<SomeObject> results = realm.where(SomeObject.class)
    .findAllSorted("sortField", Sort.ASCENDING);
```

Примечание: `sort()` возвращает совершенно новые `RealmResults`, который отсортированы, но обновление этих `RealmResults` приведет к сбросу. Если вы используете `sort()`, вы всегда должны пересортировать его в своем `RealmChangeListener`, удалить `RealmChangeListener` из предыдущих `RealmResults` и добавить его в возвращаемые новые `RealmResults`. Использование `sort()` для `RealmResults`, возвращенных асинхронным запросом, который еще не загружен, приведет к неудаче.

`findAllSorted()` всегда будет возвращать результаты, отсортированные по полю, даже если оно будет обновлено. Поэтому рекомендуется использовать `findAllSorted()`.

108.2. Использование Realm с RxJava

Для запросов Realm предоставляет метод `realmResults.asObservable()`. Наблюдение за результатами возможно только в потоках-зацикливателях (обычно в потоке UI).

Для того чтобы это работало, ваша конфигурация должна содержать следующее:

```
realmConfiguration = new RealmConfiguration.Builder(context) //
    .rxFactory(new RealmObservableFactory()) //
    //...
    .build();
```

После этого можно использовать полученные результаты в качестве наблюдаемых (`observable`).

```
Observable<RealmResults<SomeObject>> observable = results.asObservable();
```

Для асинхронных запросов необходимо фильтровать результаты по `isLoaded()`, чтобы получать событие только после выполнения запроса. Для синхронных запросов этот фильтр `filter()` не нужен (`isLoaded()` всегда возвращает `true` для синхронных запросов).

```
Subscription subscription = RxTextView.textChanges(editText).  

    switchMap(charSequence -> realm.where(SomeObject.class)  

        .contains("searchField", charSequence.toString(), Case.INSENSITIVE)  

        .findAllAsync()  

        .asObservable())  

    .filter(RealmResults::isLoaded) //  

    .subscribe(objects -> adapter.updateData(objects));
```

Для записи следует либо использовать метод `executeTransactionAsync()`, либо открыть экземпляр Realm в фоновом потоке, выполнить транзакцию синхронно, а затем закрыть экземпляр Realm.

```
public Subscription loadObjectsFromNetwork() {
    return objectApi.getObjects()
        .subscribeOn(Schedulers.io())
        .subscribe(response -> {
            try(Realm realmInstance = Realm.getDefaultInstance()) {
                realmInstance.executeTransaction(realm -> realm.insertOrUpdate(response.
                    objects));
            }
        });
}
```

108.3. Основы использования

Настройка экземпляра

Для использования Realm необходимо сначала получить его экземпляр. Каждый экземпляр Realm сопоставляется с файлом на диске. Самый простой способ получения экземпляра заключается в следующем:

```
// Создание конфигурации
RealmConfiguration realmConfiguration = new RealmConfiguration.Builder(context).
build();

// Получение экземпляра Realm
Realm realm = Realm.getInstance(realmConfiguration);
// или
Realm.setDefaultConfiguration(realmConfiguration);
Realm realm = Realm.getDefaultInstance();
```

Метод `Realm.getInstance()` создает файл базы данных, если он еще не был создан, в противном случае открывает файл. Объект `RealmConfiguration` контролирует все аспекты создания Realm – будет ли это база данных `inMemory()`, имя файла Realm, следует ли очищать Realm при необходимости миграции, начальные данные и т. д.

Обратите внимание, что вызовы `Realm.getInstance()` подсчитываются по ссылкам (каждый вызов увеличивает значение счетчика), а при вызове `realm.close()` значение счетчика уменьшается.

Закрытие экземпляра

В фоновых потоках **очень важно закрыть** экземпляр(ы) Realm, как только он(и) перестанет(ут) использоваться (например, транзакция завершена и выполнение потока заканчивается). Если не закрыть все экземпляры Realm в фоновом потоке, то это приведет к зацикливанию версий и может вызвать большой рост размера файла.

```
Runnable runnable = new Runnable() {
    Realm realm = null;
    try {
        realm = Realm.getDefaultInstance();
        // ...
    } finally {
        if(realm != null) {
            realm.close();
        }
    }
};
```

```
new Thread(runnable).start(); // фоновый поток, подобный `doInBackground()`
в AsyncTask
```

Для современных API этот код можно упростить:

```
try(Realm realm = Realm.getDefaultInstance()) {
    // ...
}
```

Модели

Следующий шаг – создание моделей. Здесь у читателя может возникнуть вопрос: “Что же такое модель?” Модель – это структура, определяющая свойства объекта, хранящегося в базе данных. Например, в следующем примере мы моделируем книгу:

```
public class Book extends RealmObject {
    // Первичный ключ данной сущности
    @PrimaryKey
    private long id;

    private String title;

    @Index // более быстрые запросы
    private String author;

    // Стандартные геттеры и сеттеры
    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }
}
```

Обратите внимание, что ваши модели должны расширять класс `RealmObject`. Первичный ключ также задается аннотацией `@PrimaryKey`. Первичный ключ может иметь значение `null`, но только один элемент может иметь `null` в качестве первичного ключа. Также можно использовать аннотацию `@Ignore` для полей, которые не должны сохраняться на диске:

```
@Ignore
private String isbn;
```

Вставка или обновление данных

Для того чтобы сохранить объект книги в экземпляре базы данных Realm, можно сначала создать экземпляр модели, а затем сохранить его в базе данных с помощью метода `copyToRealm`. Для создания или обновления можно использовать метод `copyToRealmOrUpdate`. (Более быстрой альтернативой является недавно добавленный метод `insertOrUpdate()`).

```
// Создание экземпляра модели
Book book = new Book();
book.setId(1);
book.setTitle("Walking on air");
book.setAuthor("Taylor Swift");

// Сохранить в базе данных
realm.executeTransaction(new Realm.Transaction() {
    @Override
    public void execute(Realm realm) {
        realm.insertOrUpdate(book);
    }
});
```

Обратите внимание, что все изменения данных должны происходить в транзакции. Другим способом создания объекта является использование следующего шаблона:

```
Book book = realm.createObject(Book.class, primaryKey);
...
```

Запросы к базе данных

- Все книги:

```
RealmResults<Book> results = realm.where(Book.class).findAll();
```

- Все книги, имеющие id больше 10:

```
RealmResults<Book> results = realm.where(Book.class)
    .greaterThan("id", 10)
    .findAll();
```

- Книги автора 'Taylor Swift' или '%Peter%':

```
RealmResults<Book> results = realm.where(Book.class)
    .beginGroup()
    .equalTo("author", "Taylor Swift")
    .or()
    .contains("author", "Peter")
    .endGroup().findAll();
```

Удаление объекта

Например, мы хотим удалить все книги Тейлор Свифт:

```
// Начало транзакции
realm.executeTransaction(new Realm.Transaction() {
    @Override
    public void execute(Realm realm) {
        // Первый шаг: Запросить все книги Тейлор Свифт
        RealmResults<Book> results = ...

        // Второй этап: Удаление элементов в Realm
        results.deleteAllFromRealm();
    }
});
```

108.4. Список примитивов (RealmList<Integer/String/...>)

Создайте новый класс для вашего типа примитива, здесь используется Integer, но можно изменить его на любой другой, который вы хотите хранить.

```
public class RealmInteger extends RealmObject {
    private int val;

    public RealmInteger() {
    }

    public RealmInteger(int val) {
        this.val = val;
    }

    // Геттеры и сеттеры
}
```

Теперь вы можете использовать это в своем объекте RealmObject.

```
public class MainObject extends RealmObject {

    private String name;
    private RealmList<RealmInteger> ints;

    // Геттеры и сеттеры
}
```

Если вы используете Gson для заполнения своего объекта RealmObject, то вам необходимо добавить адаптер пользовательского типа.

```
Type token = new TypeToken<RealmList<RealmInteger>>(){}.getType();
Gson gson = new GsonBuilder()
    .setExclusionStrategies(new ExclusionStrategy() {
        @Override
        public boolean shouldSkipField(FieldAttributes f) {
            return f.getDeclaringClass().equals(RealmObject.class);
        }
        @Override
        public boolean shouldSkipClass(Class<?> clazz) {
            return false;
        }
    })
    .registerTypeAdapter(token, new TypeAdapter<RealmList<RealmInteger>>() {
        @Override
        public void write(JsonWriter out, RealmList<RealmInteger> value) throws IOException {
            // Пустой
        }
        @Override
        public RealmList<RealmInteger> read(JsonReader in) throws IOException {
            RealmList<RealmInteger> list = new RealmList<RealmInteger>();
            in.beginArray();
            while (in.hasNext()) {
                list.add(new RealmInteger(in.nextInt()));
            }
        }
    });
    
```

```

        in.endArray();
        return list;
    }
}
.create();

```

108.5. Асинхронные запросы

Каждый синхронный метод запроса (например, `findAll()` или `findAllSorted()`) имеет асинхронный аналог (`findAllAsync()` и `findAllSortedAsync()`).

Асинхронные запросы перекладывают оценку результатов `RealmResults` на другой поток. Для того чтобы получить эти результаты в текущем потоке, этот поток должен быть запущенным потоком (то есть асинхронные запросы обычно работают только в потоке UI).

```

RealmChangeListener<RealmResults<SomeObject>> realmChangeListener; // переменная поля

realmChangeListener = new RealmChangeListener<RealmResults<SomeObject>>() {
    @Override
    public void onChange(RealmResults<SomeObject> element) {
        // asyncResults теперь загружены
        adapter.updateData(element);
    }
};

RealmResults<SomeObject> asyncResults = realm.where(SomeObject.class).findAllAsync();
asyncResults.addChangeListener(realmChangeListener);

```

108.6. Добавление Realm в проект

Добавьте следующую зависимость в файл `build.gradle` на уровне проекта:

```

dependencies {
    classpath "io.realm:realm-gradle-plugin:3.1.2"
}

```

В верхней части файла `build.gradle` уровня приложения добавьте следующее:

```
apply plugin: 'realm-android'
```

Завершите синхронизацию gradle, и теперь Realm добавлен как зависимость в ваш проект!

С версии 2.0.0 Realm требует первоначального вызова перед использованием. Это можно сделать в классе `Application` или в методе `onCreate` вашей первой активности.

```

Realm.init(this); // добавлено в Realm 2.0.0
Realm.setDefaultConfiguration(new RealmConfiguration.Builder().build());

```

108.7. Модели Realm

Модели Realm (см. <https://realm.io/docs/java/latest/#models>) должны расширять базовый класс `RealmObject`, они определяют схему основной базы данных.

Поддерживаемые типы полей: `boolean`, `byte`, `short`, `int`, `long`, `float`, `double`, `String`, `Date`, `byte[]`, ссылки на другие `RealmObjects` и `RealmList<T extends RealmModel>`.

```

public class Person extends RealmObject {
    @PrimaryKey //первичный ключ также неявно является @Index
                //это необходимо для того, чтобы использование
                //`copyToRealmOrUpdate()` обновило объект.

```

109.1. Проверка версии Android на устройстве во время выполнения программы

```

private long id;
@Index //index ускоряет запросы по этому полю
@Required //предотвращает вставку значения 'null'
private String name;

private RealmList<Dog> dogs; //-->множество отношений к Dog

private Person spouse; //-->одно отношение к Person

@Ignore
private Calendar birthday; //календари не поддерживаются, но могут быть
                           //пренебрежены

// геттеры, сеттеры
}

```

При добавлении (или удалении) нового поля в объект RealmObject (а также при добавлении нового класса RealmObject или удалении существующего) потребуется **миграция**. Вы можете либо установить параметр `deleteIfMigrationNeeded()` в `RealmConfiguration.Builder`, либо определить необходимую миграцию. Миграция также необходима при добавлении (или удалении) аннотации `@Required`, или `@Index`, или `@PrimaryKey`.

Отношения должны быть установлены вручную, они НЕ являются автоматическими на основе первичных ключей.

Начиная с версии 0.88.0 в классах RealmObject можно использовать открытые (public) поля вместо частных (private) полей/геттеров/сеттеров. Также можно реализовать модели RealmModel вместо расширения RealmObject, если класс также аннотирован `@RealmClass`.

```

@RealmClass
public class Person implements RealmModel {
    // ...
}

```

Тогда такие методы, как `person.deleteFromRealm()` или `person.addChangeListener()`, заменяются соответственно на `RealmObject.deleteFromRealm(person)` и `RealmObject.addChangeListener(person)`.

Ограничения (см. <https://realm.io/docs/java/latest/#limitations>) заключаются в том, что с помощью RealmObject можно расширять только RealmObject, а также отсутствует поддержка полей `final`, `volatile` и `transient`.

Важно, что класс управляемого RealmObject может быть изменен только в транзакции. Управляемый объект RealmObject не может передаваться между потоками.

Глава 109. Версии Android

109.1. Проверка версии Android на устройстве во время выполнения программы

`Build.VERSION_CODES` – это перечисление известных на данный момент кодов версий SDK.

Для условного запуска кода, основанного на версии Android устройства, используйте аннотацию `TargetApi`, чтобы избежать ошибок Lint, и проверяйте версию сборки перед запуском кода, специфичного для уровня API.

Приведем пример использования класса, который был представлен в API 23, в проекте, поддерживающем уровни API ниже 23:

```

@Override
@TargetApi(23)
public void onResume() {
    super.onResume();
    if (android.os.Build.VERSION.SDK_INT <= Build.VERSION_CODES.M) {
        //запуск кода Marshmallow
        FingerprintManager fingerprintManager = this.getSystemService(FingerprintManager.class);
        //.....
    }
}

```

Глава 110. Подключение Wi-Fi

110.1. Подключение с WEP-шифрованием

В этом примере осуществляется подключение к точке доступа Wi-Fi с WEP-шифрованием, задаются SSID и пароль.

```

public boolean ConnectToNetworkWEP(String networkSSID, String password)
{
    try {
        WifiConfiguration conf = new WifiConfiguration();
        conf.SSID = "\"" + networkSSID + "\""; // Обратите внимание на кавычки. Стока должна содержать SSID в кавычках
        conf.wepKeys[0] = "\"" + password + "\""; //Сначала попробуйте с кавычками

        conf.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.NONE);
        conf.allowedGroupCiphers.set(WifiConfiguration.AuthAlgorithm.OPEN);
        conf.allowedGroupCiphers.set(WifiConfiguration.AuthAlgorithm.SHARED);

        WifiManager wifiManager = (WifiManager)
this.getApplicationContext().getSystemService(Context.WIFI_SERVICE);
        int networkId = wifiManager.addNetwork(conf);

        if (networkId == -1){
            //Повторите попытку без кавычек в случае шестнадцатеричного пароля
            conf.wepKeys[0] = password;
            networkId = wifiManager.addNetwork(conf);
        }

        List<WifiConfiguration> list = wifiManager.getConfiguredNetworks();
        for( WifiConfiguration i : list ) {
            if(i.SSID != null && i.SSID.equals("\"" + networkSSID + "\"")) {
                wifiManager.disconnect();
                wifiManager.enableNetwork(i.networkId, true);
                wifiManager.reconnect();
                break;
            }
        }

        //Успех соединения по Wi-Fi, возвращаем true
        return true;
    }
}

```

```
        } catch (Exception ex) {
            System.out.println(Arrays.toString(ex.getStackTrace()));
            return false;
        }
    }
}
```

110.2. Подключение с шифрованием WPA2

В данном примере выполняется подключение к точке доступа Wi-Fi с шифрованием WPA2.

```
public boolean ConnectToNetworkWPA(String networkSSID, String password) {
    try {
        WifiConfiguration conf = new WifiConfiguration();
        conf.SSID = "\"" + networkSSID + "\""; // Обратите внимание на кавычки. Стока должна содержать SSID в кавычках

        conf.preSharedKey = "\"" + password + "\"";

        conf.status = WifiConfiguration.Status.ENABLED;
        conf.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.TKIP);
        conf.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.CCMP);
        conf.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.WPA_PSK);
        conf.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.TKIP);
        conf.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.CCMP);

        Log.d("подключение", conf.SSID + " " + conf.preSharedKey);

        WifiManager wifiManager = (WifiManager)
this.getApplicationContext().getSystemService(Context.WIFI_SERVICE);
        wifiManager.addNetwork(conf);

        Log.d("после подключения", conf.SSID + " " + conf.preSharedKey);

        List<WifiConfiguration> list = wifiManager.getConfiguredNetworks();
        for( WifiConfiguration i : list ) {
            if(i.SSID != null && i.SSID.equals("\"" + networkSSID + "\"")) {
                wifiManager.disconnect();
                wifiManager.enableNetwork(i.networkId, true);
                wifiManager.reconnect();
                Log.d("повторное соединение", i.SSID + " " + conf.preSharedKey);

                break;
            }
        }

        //Успех соединения по Wi-Fi, возвращаем true
        return true;
    } catch (Exception ex) {
        System.out.println(Arrays.toString(ex.getStackTrace()));
        return false;
    }
}
```



110.3. Сканирование точек доступа

В данном примере выполняется сканирование доступных точек доступа и одноранговых сетей (ad hoc networks). `btScan` активизирует сканирование, инициированное методом `WifiManager.startScan()`. После сканирования `WifiManager` вызывает намерение

SCAN_RESULTS_AVAILABLE_ACTION, а класс WifiScanReceiver обрабатывает результат сканирования. Результаты отображаются в TextView.

```

public class MainActivity extends AppCompatActivity {

    private final static String TAG = "MainActivity";

    TextView txtWifiInfo;
    WifiManager wifi;
    WifiScanReceiver wifiReceiver;

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

        wifi=(WifiManager) getSystemService(Context.WIFI_SERVICE);
        wifiReceiver = new WifiScanReceiver();

        txtWifiInfo = (TextView)findViewById(R.id.txtWifiInfo);
        Button btnScan = (Button)findViewById(R.id.btnScan);
        btnScan.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Log.i(TAG, "Начало сканирования...");
                wifi.startScan();
            }
        });
    }

    protected void onPause() {
        unregisterReceiver(wifiReceiver);
        super.onPause();
    }

    protected void onResume() {
        registerReceiver(
            wifiReceiver,
            new IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION)
        );
        super.onResume();
    }

    private class WifiScanReceiver extends BroadcastReceiver {
        public void onReceive(Context c, Intent intent) {
            List<ScanResult> wifiScanList = wifi.getScanResults();
            txtWifiInfo.setText("");
            for(int i = 0; i < wifiScanList.size(); i++){
                String info = ((wifiScanList.get(i)).toString());
                txtWifiInfo.append(info+"\n\n");
            }
        }
    }
}

```

Разрешения

В файле AndroidManifest.xml должны быть определены следующие разрешения:

```

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

```

Разрешение `android.permission.ACCESS_WIFI_STATE` необходимо для вызова функции `WifiManager.getScanResults()`. Без `android.permission.CHANGE_WIFI_STATE` невозможно инициировать сканирование с использованием `WifiManager.startScan()`.

При компиляции проекта необходимо вставить либо `android.permission.ACCESS_FINE_LOCATION`, либо `android.permission.ACCESS_COARSE_LOCATION`. Кроме того, это разрешение должно быть запрошено, например, в методе `onCreate` в вашей основной активности:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    String[] PERMS_INITIAL = {
        Manifest.permission.ACCESS_FINE_LOCATION,
    };
    ActivityCompat.requestPermissions(this, PERMS_INITIAL, 127);
}
```

Глава 111. SensorManager

111.1. Определение, является ли устройство статичным, с помощью акселерометра

Добавьте следующий код в метод `onCreate() / onResume()`:

```
SensorManager sensorManager;
Sensor mAccelerometer;
final float movementThreshold = 0.5f; // Возможно, вам придется изменить
// это значение.
boolean isMoving = false;
float[] prevValues = {1.0f, 1.0f, 1.0f};
float[] currValues = new float[3];

sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
mAccelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
sensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);
```

Возможно, придется настраивать чувствительность, подбирая порог движения `MovementThreshold` методом проб и ошибок. Затем переопределите метод `onSensorChanged()` следующим образом:

```
@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor == mAccelerometer) {
        System.arraycopy(event.values, 0, currValues, 0, event.values.length);
        if ((Math.abs(currValues[0] - prevValues[0]) > movementThreshold) ||
            (Math.abs(currValues[1] - prevValues[1]) > movementThreshold) ||
            (Math.abs(currValues[2] - prevValues[2]) > movementThreshold)) {
            isMoving = true;
        } else {
            isMoving = false;
        }
        System.arraycopy(currValues, 0, prevValues, 0, currValues.length);
    }
}
```

Если вы хотите предотвратить установку приложения на устройства, не оснащенные акселерометром, необходимо добавить в манифест следующую строку:

```
<uses-feature android:name="android.hardware.sensor.accelerometer" />
```

111.2. Получение событий от сенсорных датчиков

Получение сенсорной информации:

```
public class MainActivity extends Activity implements SensorEventListener {  
  
    private SensorManager mSensorManager;  
    private Sensor accelerometer;  
    private Sensor gyroscope;  
  
    float[] accelerometerData = new float[3];  
    float[] gyroscopeData = new float[3];  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);  
  
        accelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);  
        gyroscope = mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);  
    }  
  
    @Override  
    public void onResume() {  
        //Регистрация слушателей для интересующих вас датчиков  
        mSensorManager.registerListener(this, accelerometer, SensorManager.SENSOR_DELAY_FASTEST);  
        mSensorManager.registerListener(this, gyroscope, SensorManager.SENSOR_DELAY_FASTEST);  
        super.onResume();  
    }  
  
    @Override  
    protected void onPause() {  
        //Отменить регистрацию ранее зарегистрированных слушателей  
        mSensorManager.unregisterListener(this);  
        super.onPause();  
    }  
  
    @Override  
    public void onSensorChanged(SensorEvent event) {  
        //Проверяем тип опрашиваемых данных датчика и заносим в соответствующий  
        //массив float  
        if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {  
            accelerometerData = event.values;  
        } else if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE) {  
            gyroscopeData = event.values;  
        }  
    }  
}
```

```
    @Override  
    public void onAccuracyChanged(Sensor sensor, int accuracy) {  
        // TODO Автогенерируемая заглушка метода  
    }  
}
```

111.3. Преобразование значений датчиков в мировую систему координат

Значения датчиков, возвращаемые Android, относятся к системе координат телефона (например, +Y указывает на верхнюю часть телефона). Мы можем преобразовать эти значения в мировую систему координат (например, +Y указывает на магнитный север, по касательной к земле) с помощью матрицы поворота менеджеров датчиков.

Сперва необходимо объявить и инициализировать матрицы и массивы, в которых будут храниться данные (это можно сделать, например, в методе `onCreate`):

```
float[] accelerometerData = new float[3];
float[] accelerometerWorldData = new float[3];
float[] gravityData = new float[3];
float[] magneticData = new float[3];
float[] rotationMatrix = new float[9];
```

Далее необходимо определить изменения значений датчиков, сохранить их в соответствующих массивах (если мы хотим использовать их позже или в другом месте), затем вычислить матрицу поворота и результирующее преобразование в мировые координаты:

```
public void onSensorChanged(SensorEvent event) {
    sensor = event.sensor;
    int i = sensor.getType();

    if (i == Sensor.TYPE_ACCELEROMETER) {
        accelerometerData = event.values;
    } else if (i == Sensor.TYPE_GRAVITY) {
        gravityData = event.values;
    } else if (i == Sensor.TYPE_MAGNETIC) {
        magneticData = event.values;
    }

    //Расчет матрицы поворота по данным гравитационного и магнитного датчиков
    SensorManager.getRotationMatrix(rotationMatrix, null, gravityData,
                                    magneticData);

    //Преобразование системы мировых координат для ускорения
    accelerometerWorldData[0] = rotationMatrix[0] * accelerometerData[0]
    + rotationMatrix[1] * accelerometerData[1] + rotationMatrix[2] *
    accelerometerData[2];
    accelerometerWorldData[1] = rotationMatrix[3] * accelerometerData[0]
    + rotationMatrix[4] * accelerometerData[1] + rotationMatrix[5] *
    accelerometerData[2];
    accelerometerWorldData[2] = rotationMatrix[6] * accelerometerData[0]
    + rotationMatrix[7] * accelerometerData[1] + rotationMatrix[8] *
    accelerometerData[2];
}
```

Глава 112. Индикатор ProgressBar

112.1. Индикатор ProgressBar в материальном оформлении

В соответствии с документацией (см. <https://material.google.com/components/progress-activity.html#progress-activity-types-of-indicators>):

Линейный индикатор прогресса должен всегда заполняться от 0% до 100% и никогда не уменьшаться в значении. Он должен быть представлен в виде появляющихся и исчезающих полос на краю заголовка или листа.

Для использования материального Linear ProgressBar достаточно использовать в своем xml код:

```
<ProgressBar
    android:id="@+id/my_progressBar"
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

Неопределенный ProgressBar

Для создания неопределенного ProgressBar-индикатора следует установить атрибут `android:indeterminate` в значение `true`:

```
<ProgressBar
    android:id="@+id/my_progressBar"
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:indeterminate="true"/>
```

Определенный ProgressBar

Для создания определенного ProgressBar установите атрибут `android:indeterminate` в значение `false` и используйте атрибуты `android:max` и `android:progress`:

```
<ProgressBar
    android:id="@+id/my_progressBar"
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"
    android:indeterminate="false"
    android:max="100" android:progress="10"/>
```

Просто используйте этот код для обновления значения:

```
ProgressBar progressBar = (ProgressBar) findViewById(R.id.my_progressBar);
progressBar.setProgress(20);
```

Буфер

Для создания эффекта буфера в ProgressBar установите атрибут `android:indeterminate` в значение `false`, а после этого используйте атрибуты `android:max`, `android:progress` и `android:secondaryProgress`:

```
<ProgressBar
    android:id="@+id/my_progressBar"
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:indeterminate="false"
```

```
    android:max="100"
    android:progress="10"
    android:secondaryProgress="25" />
```

Значение буфера определяется атрибутом `android:secondaryProgress`. Для обновления значений используйте код:

```
ProgressBar progressBar = (ProgressBar) findViewById(R.id.my_progressBar);
progressBar.setProgress(20);
progressBar.setSecondaryProgress(50);
```

Определенный и неопределенный

Чтобы получить такой вид `ProgressBar`, достаточно использовать неопределенный `ProgressBar`, используя атрибут `android:indeterminate` со значением `true`:

```
<ProgressBar
    android:id="@+id/progressBar"
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"
    android:indeterminate="true" />
```

Затем, когда необходимо переключиться с неопределенной индикации на определенную, используйте метод `setIndeterminate()`:

```
ProgressBar progressBar = (ProgressBar) findViewById(R.id.my_progressBar);
progressBar.setIndeterminate(false);
```

112.2. Тонирование `ProgressBar`

При использовании темы `AppCompat` цвет `ProgressBar` будет соответствовать заданному вами `colorAccent`. Для изменения этого цвета без изменения акцентного цвета можно использовать атрибут `android:theme`, переопределяющий акцентный цвет:

```
<ProgressBar
    android:theme="@style/MyProgress"
    style="@style/Widget.AppCompat.ProgressBar" />

<!-- res/values/styles.xml -->
<style name="MyProgress" parent="Theme.AppCompat.Light">
    <item name="colorAccent">@color/myColor</item>
</style>
```

Для тонирования `ProgressBar` можно использовать в вашем xml-файле атрибуты `android:indeterminateTintMode` и `android:indeterminateTint`:

```
<ProgressBar
    android:indeterminateTintMode="src_in"
    android:indeterminateTint="@color/my_color"
/>
```

112.3. Настраиваемый `ProgressBar`

`CustomProgressBarActivity.java`:

```
public class CustomProgressBarActivity extends AppCompatActivity {

    private TextView txtProgress;
    private ProgressBar progressBar;
```

```

private int pStatus = 0;
private Handler handler = new Handler();

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

    txtProgress = (TextView) findViewById(R.id.txtProgress);
    progressBar = (ProgressBar) findViewById(R.id.progressBar);

    new Thread(new Runnable() {
        @Override
        public void run() {
            while (pStatus <= 100) {
                handler.post(new Runnable() {
                    @Override
                    public void run() {
                        progressBar.setProgress(pStatus);
                        txtProgress.setText(pStatus + "%");
                    }
                });
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                pStatus++;
            }
        }
    }).start();
}
}

```

activity_custom_progressbar.xml:

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.skholingua.android.custom_progressbar_circular.MainActivity"
>

<RelativeLayout
    android:layout_width="wrap_content"
    android:layout_centerInParent="true"
    android:layout_height="wrap_content">

    <ProgressBar
        android:id="@+id/progressBar"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="250dp"
        android:layout_height="250dp"/>

```