

```
<string name="hello_world">Hello World!</string>
</resources>
```

После того как строка определена в файле ресурсов XML, она может использоваться другими частями приложения. В XML-файлах проекта приложения можно использовать элемент `<string>`, обращаясь к `@string/string_name`. Например, файл манифеста приложения (`/manifests/AndroidManifest.xml`) по умолчанию содержит следующую строку:

```
android:label="@string/app_name"
```

Этот код указывает Android на необходимость поиска ресурса `<string>` под названием `"app_name"` для использования в качестве имени приложения при его установке или отображении в запуске.

Еще одним случаем использования ресурса `<string>` из XML-файла в Android является использование его в файле компоновки. Например, ниже представлено `TextView`, отображающее строку `hello_world`, которую мы определили ранее:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world"/>
```

Вы также можете получить доступ к ресурсам `<string>` из java-части вашего приложения. Чтобы вызвать нашу строку `hello_world`, описанную выше, в классе `Activity`, используйте:

```
String helloWorld = getString(R.string.hello_world);
```

38.5. Определение размеров

Размеры обычно хранятся в файле ресурсов с именем `dimens.xml`. Они определяются с помощью элемента `<dimen>`.

`res/values/dimens.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="small_padding">5dp</dimen>
    <dimen name="medium_padding">10dp</dimen>
    <dimen name="large_padding">20dp</dimen>
    <dimen name="small_font">14sp</dimen>
    <dimen name="medium_font">16sp</dimen>
    <dimen name="large_font">20sp</dimen>
</resources>
```

Можно использовать различные единицы измерения:

- `sp` – масштабно-независимые пиксели (Scale-independent Pixels); для шрифтов
- `dp` – пиксели, не зависящие от плотности (Density-independent Pixels); для всего остального
- `pt` – пункты (points)
- `px` – пиксели
- `mm` – миллиметры
- `in` – дюймы

Теперь на измерения можно ссылаться в XML с помощью синтаксиса `@dimen/name_of_the_dimension`. Например:

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/large_padding">
</RelativeLayout>
```

38.6. Форматирование строк в файле strings.xml

Определение строк в файле strings.xml также позволяет форматировать строки. Единственная оговорка заключается в том, что со строкой необходимо будет работать в коде, как показано ниже, а не просто прикреплять ее к макету:

```
<string name="welcome_trainer">Здравствуйте, тренер покемонов, %1$s! Вы поймали
%2$d покемонов.</string> String welcomePokemonTrainerText = getString(R.string.
welcome_trainer, tranerName, pokemonCount);
```

Рассмотрим %1\$s в приведенном примере поэлементно:

‘%’ отделяет от обычных символов,

‘1’ обозначает первый параметр,

‘\$’ используется в качестве разделителя между номером и типом параметра,

‘s’ обозначает строковый тип (‘d’ используется для целого числа)

Обратите внимание, что getString() является методом Context или Resources, то есть его можно использовать непосредственно внутри экземпляра Activity, либо использовать getActivity().getString() или getContext().getString() соответственно.

38.7. Определение целочисленного массива

Для определения целочисленного массива запишите в файл ресурсов

res/values/filename.xml

```
<integer-array name="integer_array_name">
    <item>integer_value</item>
    <item>@integer/integer_id</item>
</integer-array>
```

например

res/values/arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <integer-array name="fibo">
        <item>@integer/zero</item>
        <item>@integer/one</item>
        <item>@integer/one</item>
        <item>@integer/two</item>
        <item>@integer/three</item>
        <item>@integer/five</item>
    </integer-array>
</resources>
```

...и используйте его из Java. Пример:

```
int[] values = getResources().getIntArray(R.array.fibo);
Log.i("TAG", Arrays.toString(values));
```

Результат:

I/TAG: [0, 1, 1, 2, 3, 5]

38.8. Определение списка состояний цвета

Такие списки состояний могут использоваться как цвета, но будут меняться в зависимости от состояния представления, для которого они используются. Чтобы определить такой список, создайте файл ресурсов в `res/color/foo.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:color="#888888" android:state_enabled="false"/>
    <item android:color="@color/lightGray" android:state_selected="false"/>
    <item android:color="@android:color/white" />
</selector>
```

Элементы оцениваются в порядке их определения, и первым используется элемент, чьи заданные состояния совпадают с текущим состоянием представления. Поэтому хорошей практикой является указание в конце элемента `catch-all`, без указания селекторов состояний.

Каждый элемент может либо использовать литерал цвета, либо ссылаться на цвет, определенный в другом месте.

38.9. 9 патчей (9-patch)

9 патчей – это **растягиваемые** изображения, в которых области, которые можно растягивать, определяются черными маркерами на прозрачной границе. Учебные материалы по теме можно найти здесь: <http://radleymarx.com/blog/simple-guide-to-9-patch/>.

Простое руководство по 9-patch

Многие Android-разработчики высказывают нарекания, что 9-patch (он же 9.png) является запутанным и плохо документированным. Поэтому необходимо хорошее руководство, чтобы разобраться в этой методике.

По сути, 9-patch использует прозрачность png для выполнения усовершенствованного варианта технологий 9-slice или scale9. Направляющие – это прямые 1-пиксельные черные линии, отрисованные на краю изображения, которые определяют масштаб и заливку изображения. Назав файл изображения `name.9.png`, Android распознает формат 9.png и будет использовать черные направляющие для масштабирования и заливки растровых изображений.

Вот основная карта-путеводитель:

9-patch guides

what do they do?

scalable area

scalable area

fill area

fill area



Как видно, на каждой стороне изображения имеются направляющие. Верхняя и левая направляющие служат для масштабирования изображения (например, 9-фрагментного), а правая и нижняя определяют область заливки.

Черные направляющие линии обрезаются/удаляются из изображения – они не будут отображаться в приложении. Направляющие должны быть шириной не более одного пикселя,

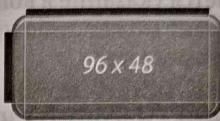
поэтому если вы хотите кнопку размером 48×48, то ваш png будет иметь размер 50×50. Все, что толще одного пикселя, будет оставаться частью изображения. (В примерах направляющие имеют ширину 4 пикселя для большей наглядности. На самом деле они должны быть только 1-пиксельными).

Направляющие должны иметь сплошной черный цвет (#000000). Даже небольшое различие в цвете (#000001) или альфа-диапазоне приведет к сбою.

Также следует помнить, что оставшаяся область однопиксельного контура должна быть полностью прозрачной. Это относится и к четырем углам изображения – они всегда должны быть прозрачными. Это может быть более серьезной проблемой, чем кажется. Например, при масштабировании изображения в Photoshop добавляются пиксели со слаживанием, которые могут включать почти невидимые пиксели. При необходимости масштабирования в Photoshop используйте настройку Nearest Neighbor в выпадающем меню Resample Image (внизу всплывающего меню Image Size), чтобы сохранить четкие края направляющих.

Scalable Area

48x48 button can be scaled to any size larger than 48x48



Верхняя и левая направляющие используются для определения масштабируемой части изображения – левая для масштабирования высоты, верхняя для масштабирования ширины. На примере изображения кнопки это означает, что кнопка может растягиваться по горизонтали и вертикали в пределах черной части, а все остальное, например, углы, останется прежнего размера. Это позволяет создавать кнопки, которые могут масштабироваться до любого размера и сохранять единый вид.

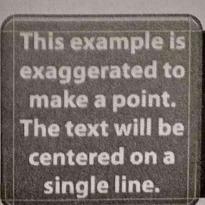
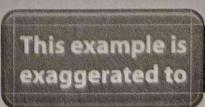
Важно отметить, что 9-патчевые изображения не уменьшаются, а только увеличиваются. Поэтому лучше всего начинать с как можно меньшего размера.

Кроме того, можно не выделять участки в середине масштабной линии. Например, если у вас есть кнопка с резким глянцевым краем посередине, вы можете опустить несколько пикселей в середине левой направляющей LEFT. Центральная горизонтальная ось изображения не будет масштабироваться, только части над и под ней, поэтому резкий блеск не будет слажен или размыт.

Fill Area



*Fill area is for button label.
Text is a single line by default,
but can be two lines or more.*



Направляющие области заливки являются необязательными и позволяют определить область для таких элементов, как текстовый ярлык. Заполнение определяет, сколько места в изображении есть для размещения текста, значка или других элементов. 9-patch подходит не только для кнопок, но и для фоновых изображений.

Приведенный выше пример кнопки и метки преувеличен просто для того, чтобы объяснить идею заливки. Честно говоря, сложно предсказать, как Android будет работать с многострочными метками, поскольку метка кнопки обычно представляет собой одну строку текста.

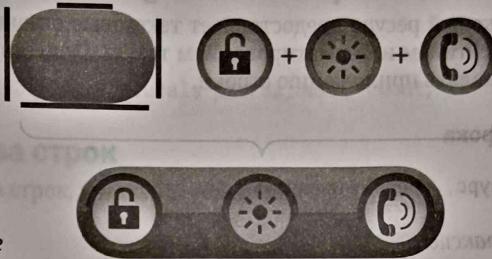
Наконец, вот хорошая демонстрация того, как могут меняться направляющие масштаба и заливки, например, LinearLayout с фоновым изображением и полностью закругленными сторонами:

Scale & Fill

for rounded sides



*Left scale is not used,
so height remains the same.
Fill guides extend close to the
ends to make room for fitted icons.*



В данном примере левая направляющая не используется, но ее наличие все равно необходимо. Фоновое изображение не масштабируется по вертикали, оно просто масштабируется по горизонтали (на основе положения верхней направляющей). Если посмотреть на направляющие заливки, то правая и нижняя направляющие выходят за границы изогнутых краев изображения. Это позволяет расположить круглые кнопки близко к краям фона, что обеспечивает плотное прилегание.

Вот и все. Вы убедитесь, что 9 патчей очень просты, как только их освоите. Это не идеальный способ масштабирования, но заполнение области и многострочные направляющие масштаба обеспечивают большую гибкость, чем традиционные 9-slice и scale9. Попробуйте, и вы быстро во всем разберетесь.

38.10. Получение ресурсов без предупреждений «deprecated»

При использовании Android API 23 и выше очень часто можно наблюдать такую ситуацию:

```
context.getResources().getColor(R.color.colorPrimaryDark);
init();
```

'getColor(int)' is deprecated more... (Ctrl+F1)

Данная ситуация вызвана структурным изменением Android API в части получения ресурсов. Теперь надо использовать следующую функцию:

```
public int getColor(@ColorRes int id, @Nullable Theme theme) throws NotFoundException
```

Однако у библиотеки android.support.v4 есть другое решение. Добавьте в свой файл build.gradle следующую зависимость:

```
com.android.support:support-v4:24.0.0
```

Тогда становятся доступными все методы из библиотеки поддержки:

```
ContextCompat.getColor(context, R.color.colorPrimaryDark);
```

```
ContextCompat.getDrawable(context, R.drawable.btn_check);
ContextCompat.getColorStateList(context, R.color.colorPrimary);
DrawableCompat.setTint(drawable);
ContextCompat.getColor(context, R.color.colorPrimaryDark));
```

Кроме того, могут быть использованы и другие методы из библиотеки поддержки:

```
ViewCompat.setElevation(textView, 1F);
ViewCompat.animate(textView);
TextViewCompat.setTextAppearance(textView, R.style.AppThemeTextStyle);
...
```

38.11. Работа с файлом strings.xml

Строковый ресурс предоставляет текстовые строки для вашего приложения с дополнительным стилем и форматированием текста. Существует три типа ресурсов, которые могут предоставлять приложению строки.

1. Страна

XML-ресурс, предоставляющий одну строку.

Синтаксис:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="string_name">text_string</string>
</resources>
```

И используйте эту строку в макете:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/string_name" />
```

2. Массив строк

XML-ресурс, предоставляющий массив строк.

Синтаксис:

```
<resources>
<string-array name="planets_array">
    <item>Меркурий</item>
    <item>Venus</item>
    <item>Земля</item>
    <item>Марс</item>
</string-array>
```

Использование:

```
Resources res = getResources();
String[] planets = res.getStringArray(R.array.planets_array);
```

3. Строки количества (множественные)

XML-ресурс, содержащий разные строки для множественности.

Синтаксис:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals>
        <item
            name="plural_name">
            <item
                quantity=["zero" | "one" | "two" | "few" | "many" | "other"]>
                <text_string></item>
            </item>
        </plurals>
    </resources>
```

Использование:

```
int count = getNumberOfsongsAvailable();
Resources res = getResources();
String songsFound = res.getQuantityString(R.plurals.plural_name, count, count);
```

38.12. Определение массива строк

Для того чтобы определить массив строк, внесите в файл ресурсов:

```
res/values/filename.xml
<string-array name="string_array_name">
    <item>text_string</item>
    <item>@string/string_id</item>
</string-array>
```

например

res/values/arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="string_array_example">
        <item>@string/app_name</item>
        <item>@string/hello_world</item>
    </string-array>
</resources>
```

...и используйте это из Java, например:

```
String[] strings = getResources().getStringArray(R.array.string_array_example);
Log.i("TAG", Arrays.toString(strings));
```

Результат:

```
I/TAG: [HelloWorld, Hello World!]
```

38.13. Определение целых чисел

Целые числа обычно хранятся в файле ресурсов с именем integers.xml, но имя файла может быть выбрано произвольно. Каждое целое число определяется с помощью элемента <integer>, как показано в следующем файле:

res/values/integers.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <integer name="max">100</integer>
</resources>
```

Теперь на целые числа можно ссылаться в XML с помощью синтаксиса @integer/name_of_the_integer, как показано в следующем примере:

```
<ProgressBar
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:max="@integer/max"/>
```

38.14. Определение ресурса меню и его использование внутри активности или фрагмента

Определим меню в res/menu:

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

    <item
        android:id="@+id/first_item_id"
        android:orderInCategory="100"
        android:title="@string/first_item_string"
        android:icon="@drawable/first_item_icon"
        app:showAsAction="ifRoom"/>

    <item
        android:id="@+id/second_item_id"
        android:orderInCategory="110"
        android:title="@string/second_item_string"
        android:icon="@drawable/second_item_icon"
        app:showAsAction="ifRoom"/>

</menu>
```

О дополнительных возможностях настройки можно прочитать здесь: <https://developer.android.com/guide/topics/resources/menu-resource.html>.

В активности:

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    // Переопределение ресурса меню
    inflater.inflate(R.menu.menu_resource_id, menu);
    super.onCreateOptionsMenu(menu, inflater);
}

@Override
public void onPrepareOptionsMenu(Menu menu) {
    // Переопределение для подготовки элементов (установка видимости, изменение текста,
    // изменение значка...)
    super.onPrepareOptionsMenu(menu);
}
```

```

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Переопределение для работы с элементами
    int menuItemId = item.getItemId();
    switch (menuItemId) {
        case R.id.first_item_id:
            return true; // возвращается true, если обработка выполнена
    }
    return super.onOptionsItemSelected(item);
}

```

Для вызова вышеуказанных методов во время показа представления следует вызвать `getActivity().invalidateOptionsMenu();`

Внутри фрагмента необходим еще один вызов:

```

@Nullable
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) { setHasOptionsMenu(true);
    super.onCreateView(inflater, container, savedInstanceState);
}

```

Глава 39. Библиотека привязки данных

39.1. Базовая привязка текстовых полей

Конфигурация Gradle (Module:app)

```

android {
    ...
    dataBinding {
        enabled = true
    }
    public void onButtonClick(User user);
}

```

Модель данных

```

public class Item {
    public String name;
    public String description;

    public Item(String name, String description) {
        this.name = name;
        this.description = description;
    }
    public String getName() {
        return name;
    }
}

```

Макет XML

Первый шаг – обернуть макет в тег `<layout>`, добавить элемент `<data>` и элемент `<variable>` для вашей модели данных.

Затем можно привязать XML-атрибуты к полям в модели данных с помощью конструкции `@{model.fieldname}`, где `model` – имя переменной, а `fieldname` – поле, к которому нужно получить доступ.

item_detail_activity.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <variable name="item" type="com.example.Item"/>
    </data>

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

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{item.name}"/>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{item.description}"/>

    </LinearLayout>
</layout>
```

Для каждого XML-файла макета, правильно сконфигурированного с привязками, плагин Android Gradle генерирует соответствующий класс `bindings`. Поскольку у нас есть макет с именем `item_detail_activity`, соответствующий генерированный класс привязки называется `ItemDetailActivityBinding`.

Затем это связывание можно использовать в активности следующим образом:

```
public class ItemDetailActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ItemDetailActivityBinding binding = DataBindingUtil.setContentView(this,
R.layout.item_detail_activity);
        Item item = new Item("Example item", "This is an example item.");
        binding.setItem(item);
    }
}
```

39.2. Встроенное двустороннее связывание данных

Двустороннее связывание данных поддерживает следующие атрибуты:

Элемент Свойства

<code>AbsListView</code>	<code>android:selectedItemPosition</code>
<code>CalendarView</code>	<code>android:date</code>
<code>CompoundButton</code>	<code>android:checked</code>
<code>DatePicker</code>	<code>android:year</code> <code>android:month</code> <code>android:day</code>
<code>EditText</code>	<code>android:text</code>

Элемент Свойства

NumberPicker	android:value
RadioGroup	android:checkedButton
RatingBar	android:rating
SeekBar	android:progress
TabHost	android:currentTab
TextView	android:text
TimePicker	android:hour android:minute
ToggleButton	android:checked
Switch	android:checked

Использование

```
<layout ...>
  <data>
    <variable type="com.example.myapp.User" name="user"/>
  </data>
  <RelativeLayout ...>
    <EditText android:text="@{user.firstName}" .../>
  </RelativeLayout>
</layout>
```

Обратите внимание, что выражение связывания Binding `@{}` содержит дополнительный символ `=`, который необходим для двустороннего связывания. Использование методов в выражениях двустороннего связывания невозможно.

39.3. Пользовательское событие с использованием лямбда-выражения

Определим интерфейс:

```
public interface ClickHandler {
  public void onButtonClick(User user);
}
```

Создадим класс Model:

```
public class User {
  private String name;

  public User(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }
}
```

Макет XML:

```

<layout xmlns:android="http://schemas.android.com/apk/res/android">

    <data>
        <variable
            name="handler"
            type="com.example.ClickHandler"/>
        <variable
            name="user"
            type="com.example.User"/>
    </data>

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

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{user.name}"
            android:onClick="@{() -> handler.onButtonClick(user)}" />
    </RelativeLayout>
</layout>

```

Код активности:

```

public class MainActivity extends Activity implements ClickHandler {

    private ActivityMainBinding binding;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = DataBindingUtil.setContentView(this, R.layout.activity_main);
        binding.setUser(new User("DataBinding User"));
        binding.setHandler(this);
    }

    @Override
    public void onButtonClick(User user) {
        Toast.makeText(MainActivity.this, "Welcome " + user.getName(), Toast.LENGTH_LONG).show();
    }
}

```

Для некоторых слушателей представлений, которые недоступны в xml-коде, но могут быть заданы в Java-коде, можно осуществить связывание с помощью пользовательской привязки.

Пользовательский класс

```

public class BindingUtil {
    @BindingAdapter({"bind:autoAdapter"})
    public static void setAdapter(AutoCompleteTextView view, ArrayAdapter<String> pArrayAdapter) {
        view.setAdapter(pArrayAdapter);
    }
    @BindingAdapter({"bind:onKeyListener"})

```

39.4. Значение по умолчанию в привязке данных

```
public static void setOnKeyListener(AutoCompleteTextView view, View.OnKeyListener pOnKeyListener) {
{
    view.setOnKeyListener(pOnKeyListener);
}
}
```

Класс обработчика

```
public class Handler extends BaseObservable {
private ArrayAdapter<String> roleAdapter;

public ArrayAdapter<String> getRoleAdapter() {
    return roleAdapter;
}

public void setRoleAdapter(ArrayAdapter<String> pRoleAdapter) {
    roleAdapter = pRoleAdapter;
}
}
```

XML

```
<layout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:bind="http://schemas.android.com/tools">

    <data>
        <variable
            name="handler"
            type="com.example.Handler" />
    </data>

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

        <AutoCompleteTextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:singleLine="true"
            bind:autoAdapter="@{handler.roleAdapter}" />
    </LinearLayout>
</layout>
```

39.4. Значение по умолчанию в привязке данных

На панели предварительного просмотра Preview отображаются значения по умолчанию для выражений привязки данных, если они предусмотрены.

Например:

```
android:layout_height="@{@dimen/main_layout_height, default=wrap_content}"
```

При проектировании он будет принимать значение wrap_content, а в панели предварительного просмотра будет действовать как wrap_content. Другой пример:

```
android:text="@{user.name, default='Preview Text'}"
```

Он отобразит текст предварительного просмотра (Preview Text) в панели предварительного просмотра, но при запуске в устройстве или эмуляторе будет отображен реальный текст, привязанный к нему.

39.5. Привязка данных в диалоге

```
public void DoSomething() {
    DialogTestBinding binding = DataBindingUtil
        .inflate(LayoutInflater.from(context), R.layout.dialog_test, null, false);

    Dialog dialog = new Dialog(context);
    dialog.setContentView(binding.getRoot());
    dialog.show();
}
```

39.6. Привязка с помощью метода доступа

Если в модели имеются частные методы, то библиотека привязки к базе данных все равно позволяет обращаться к ним в представлении без использования полного имени метода.

Модель данных

```
public class Item {
    private String name;

    public String getName() {
        return name;
    }
}
```

Макет XML

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <variable name="item" type="com.example.Item"/>
    </data>

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <!-- Поскольку в нашей модели данных поле "name" является частным,
        при таком связывании вместо него будет использоваться открытый метод
        getName(). -->
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{item.name}"/>
    </LinearLayout>
</layout>
```

39.7. Передача виджета в качестве ссылки в BindingAdapter

Макет XML:

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

```

<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        </data>
        <LinearLayout
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_height="match_parent">
            <ProgressBar
                android:id="@+id/progressBar"
                style="?android:attr/progressBarStyleSmall"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"/>
            <ImageView
                android:id="@+id/img"
                android:layout_width="match_parent"
                android:layout_height="100dp"
                app:imageUrl="@{url}"
                app:progressbar="@{progressBar}"/>
        </LinearLayout>
    </layout>

```

Метод BindingAdapter:

```

@BindingAdapter({"imageUrl", "progressbar"})
public static void loadImage(ImageView view, String imageUrl, ProgressBar
progressBar) {
    Glide.with(view.getContext()).load(imageUrl)
        .listener(new RequestListener<String, GlideDrawable>() {
            @Override
            public boolean onException(Exception e, String model,
Target<GlideDrawable> target, boolean isFirstResource) {
                return false;
            }
            @Override(ListItemBinding binding)
            public boolean onResourceReady(GlideDrawable resource, String model,
Target<GlideDrawable> target, boolean isFromMemoryCache, boolean
isFirstResource) {
                progressBar.setVisibility(View.GONE);
                return false;
            }
        }).into(view);
}

```

39.8. Слушатель кликов с привязкой

Создание интерфейса для clickHandler

```

public interface ClickHandler {
    public void onButtonClick(View v);
}

```

Макет XML

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

    <data>
        <variable
            name="handler"
            type="com.example.ClickHandler"/>
    </data>

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

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="click me"
            android:onClick="@{handler.onButtonClick}"/>
    </RelativeLayout>
</layout>
```

Обработка событий в вашей активности

```
public class MainActivity extends Activity implements ClickHandler {

    private ActivityMainBinding binding;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = DataBindingUtil.setContentView(this,R.layout.activity_main);
        binding.setHandler(this);
    }

    @Override
    public void onButtonClick(View v) {
        Toast.makeText(context, "Button clicked",Toast.LENGTH_LONG).show();
    }
}
```

39.9. Привязка данных в адаптере RecyclerView Adapter

Также можно использовать привязку данных внутри адаптера RecyclerView.

Модель данных

```
public class Item {
    private String name;

    public String getName() {
        return name;
    }
}
```

Макет XML

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

```
    android:layout_height="wrap_content"
    android:text="@{item.name}"/>
```

Класс адаптера

```
public class ListItemAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder> {

    private Activity host;
    private List<Item> items;

    public ListItemAdapter(Activity activity, List<Item> items) {
        this.host = activity;
        this.items = items;
    }

    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        // раздуваем макет и получаем привязку
        ListItemBinding binding = DataBindingUtil.inflate(host.getLayoutInflater()
            R.layout.list_item, parent, false);
        return new ItemViewHolder(binding);
    }

    @Override
    public void onBindViewHolder(RecyclerView.ViewHolder holder, int position) {
        Item item = items.get(position);

        ItemViewHolder itemViewHolder = (ItemViewHolder)holder;
        itemViewHolder.bindItem(item);
    }

    @Override
    public int getItemCount() {
        return items.size();
    }

    private static class ItemViewHolder extends RecyclerView.ViewHolder {
        ListItemBinding binding;
        ItemViewHolder(ListItemBinding binding) {
            super(binding.getRoot());
            this.binding = binding;
        }

        void bindItem(Item item) {
            binding.setItem(item);
            binding.executePendingBindings();
        }
    }
}
```

39.10. Привязка данных во фрагменте

Модель данных

```
public class Item {
    private String name;
```

```

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}
}

```

Макет XML

```

<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <variable name="item" type="com.example.Item"/>
    </data>

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

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{item.name}"/>

    </LinearLayout>
</layout>

```

Фрагмент

```

@Override
public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container,
@Nullable Bundle savedInstanceState) {
    FragmentTest binding = DataBindingUtil.inflate(inflater, R.layout.fragment_test,
        container, false);
    Item item = new Item();
    item.setName("Thomas");
    binding.setItem(item);
    return binding.getRoot();
}

```

39.11. Привязывание данных с пользовательскими переменными типов int, boolean

Иногда нам необходимо выполнить базовые операции, например, скрыть или показать представление на основе одного значения, для которого мы не можем создать модель, или создавать модель для этого нецелесообразно. Привязывание данных поддерживает основные типы данных для выполнения таких операций:

```

<layout xmlns:android="http://schemas.android.com/apk/res/android">

    <data>

        <import type="android.view.View" />

```

```

<variable
    name="selected"
    type="Boolean" />

</data>

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World"
        android:visibility="@{selected ? View.VISIBLE : View.GONE}" />

</RelativeLayout>
</layout>

```

Установим его значение из класса Java:

```
binding.setSelected(true);
```

39.12. Ссылочные классы

Модель данных

```

public class Item {
    private String name;

    public String getName() {
        return name;
    }
}

```

Макет XML

Вы должны импортировать ссылочные классы, как это делается в Java:

```

<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <import type="android.view.View"/>
        <variable name="item" type="com.example.Item"/>
    </data>

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

        <!-- Мы ссылаемся на класс View, чтобы установить видимость этого TextView -->
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{item.name}" />
    
```

```

        android:visibility="@{item.name == null ? View.VISIBLE : View.GONE}"/>
    </LinearLayout>
</layout>

```

Примечание: пакет `java.lang.*` импортируется системой автоматически (то же самое делает JVM для Java).

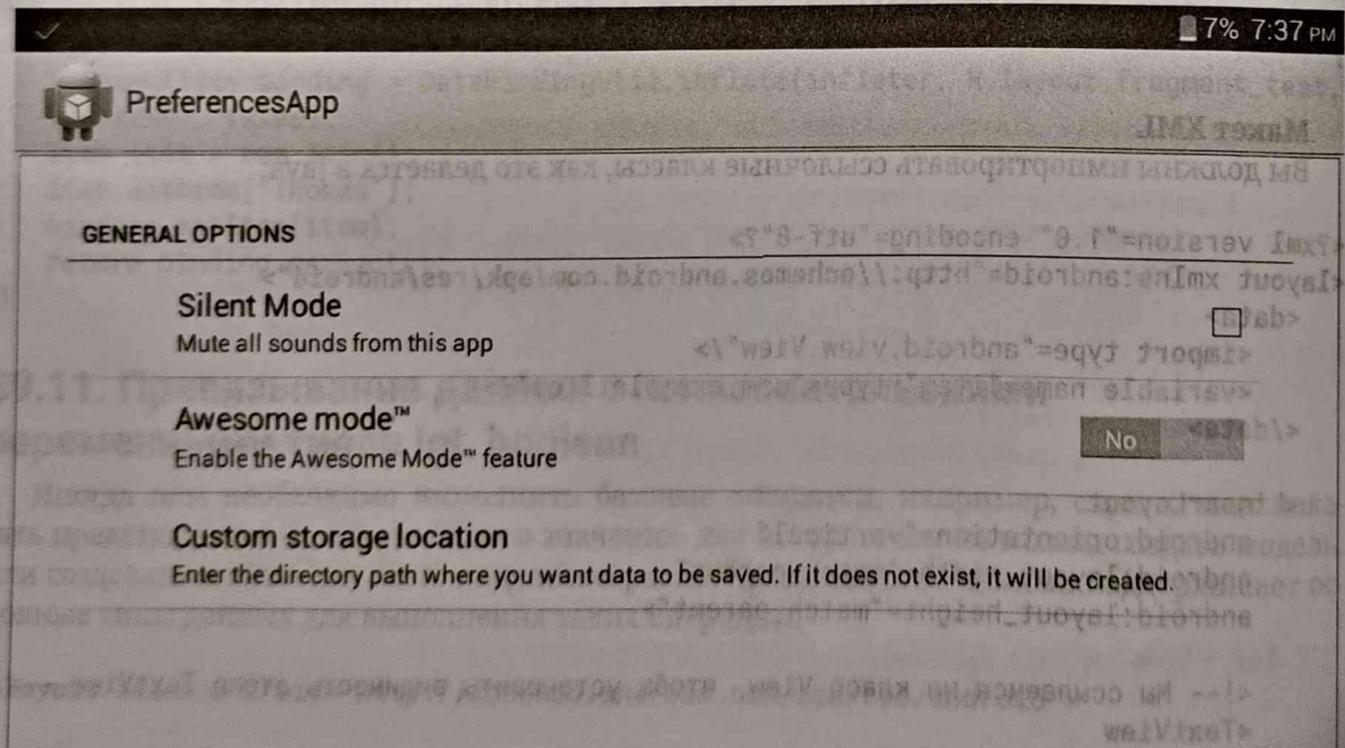
Глава 40. SharedPreferences

Параметр	Подробности
<code>key</code>	Ненулевая строка, идентифицирующая параметр. Она может содержать пробелы или непечатаемые символы. Используется только внутри приложения (и в XML-файле), поэтому его не обязательно размещать в пространстве имен, но неплохо иметь в качестве константы в исходном коде. Не локализуйте его.
<code>defValue</code>	Все функции <code>get</code> принимают значение по умолчанию, которое возвращается, если заданный ключ отсутствует в <code>SharedPreferences</code> . Оно не возвращается, если ключ присутствует, но значение имеет неправильный тип: в этом случае возникает исключение <code>ClassCastException</code> .

`SharedPreferences` предоставляет возможность сохранения данных на диске в виде пар «ключ-значение».

40.1. Реализация экрана настроек с помощью SharedPreferences

Одним из вариантов использования `SharedPreferences` является реализация в приложении экрана «Настройки», на котором пользователь может задать свои предпочтения или опции. Например:



Окно `PreferenceScreen` сохраняет пользовательские настройки в `SharedPreferences`. Для создания этого окна необходимо иметь следующее:

XML-файл для определения доступных опций

Это файл /res/xml/preferences.xml, и для приведенного выше экрана настроек он выглядит следующим образом:

```

<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory
        android:title="Общие параметры">
        <CheckBoxPreference android:key = "silent_mode"
            android.defaultValue="false"
            android:title="Silent Mode"
            android:summary="Mute all sounds from this app" />

        <SwitchPreference
            android:key="awesome_mode"
            android.defaultValue="false"
            android.switchTextOn="Yes"
            android.switchTextOff="No"
            android:title="Awesome mode™"
            android:summary="Enable the Awesome Mode™ feature" />

        <EditTextPreference
            android:key="custom_storage"
            android.defaultValue="/sdcard/data/"
            android:title="Custom storage location"
            android:summary="Enter the directory path where you want data to be
            saved. If it does not exist, it will be created."
            android:dialogTitle="Enter directory path (eg. /sdcard/data/)" />
    </PreferenceCategory>
</PreferenceScreen>
```

Это определяет доступные опции на экране настроек. Существует множество других типов предпочтений, перечисленных в документации Android по классу предпочтений (Preference Class, см. <https://developer.android.com/reference/android/preference/Preference.html>).

Далее нам понадобится **активность для размещения пользовательского интерфейса Preferences**:

```

package com.example.preferences;

import android.preference.PreferenceActivity;
import android.os.Bundle;

public class PreferencesActivity extends PreferenceActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.preferences);
    }
}
```

Она расширяет PreferenceActivity и предоставляет пользовательский интерфейс для экрана предпочтений. Ее можно запустить так же, как и обычную активность, в данном случае с помощью кода наподобие:

```
Intent i = new Intent(this, PreferencesActivity.class);
startActivity(i);
```

Не забудьте добавить `PreferencesActivity` в ваш файл `AndroidManifest.xml`.

Получить значения предпочтений внутри вашего приложения довольно просто, достаточно сначала вызвать `setDefaultValues()`, чтобы установить значения по умолчанию, определенные в вашем XML, а затем получить значения по умолчанию `SharedPreferences`. Пример:

```
//установить значения по умолчанию, которые мы определили в XML
PreferenceManager.setDefaultValues(this, R.xml.preferences, false);
SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(this);

//получение значений параметров настройки
boolean silentMode = preferences.getBoolean("silent_mode", false);
boolean awesomeMode = preferences.getBoolean("awesome_mode", false);

String customStorage = preferences.getString("custom_storage", "");
```

40.2. Commit и Apply

Метод `editor.apply()` является **асинхронным**, а `editor.commit()` – **синхронным**. Поэтому очевидно, что следует вызывать либо `apply()`, либо `commit()`:

```
SharedPreferences settings = getSharedPreferences(PREFS_FILE, MODE_PRIVATE);
SharedPreferences.Editor editor = settings.edit(); editor.putBoolean(PREF_CONST,
true);
// Это позволит асинхронно сохранить общие предпочтения без удержания текущего потока.
editor.apply();

SharedPreferences settings = getSharedPreferences(PREFS_FILE, MODE_PRIVATE);
SharedPreferences.Editor editor = settings.edit();
editor.putBoolean(PREF_CONST, true);
// Это синхронно сохранит общие предпочтения, удерживая текущий поток до завершения,
и возвратит флаг успеха
boolean result = editor.commit();
```

Функция `apply()` выполняет фиксацию без возврата булевой величины, указывающей на успех или неудачу. Функция `commit()` возвращает `true`, если сохранение работает, `false` – в противном случае.

Функция `apply()` была добавлена, так как команда разработчиков Android заметила, что практически никто не обращает внимания на возвращаемое значение, поэтому `apply` работает быстрее, так как является асинхронной.

В отличие от `commit()`, которая синхронно записывает свои предпочтения в постоянное хранилище, `apply()` немедленно фиксирует свои изменения в памяти `SharedPreferences`, но начинает асинхронную фиксацию на диск, и вы не будете уведомлены о сбоях. Если другой редактор на этих `SharedPreferences` выполнит обычную фиксацию `commit()`, в то время как `apply()` еще не завершена, `commit()` будет блокироваться до тех пор, пока не будут завершены все асинхронные фиксации (`apply`), а также все другие синхронные фиксации, которые могут быть еще не завершены.

40.3. Чтение и запись значений в SharedPreferences

```
public class MyActivity extends Activity {

    private static final String PREFS_FILE = "NameOfYourPreferenceFile";
    // PREFS_MODE определяет, какие приложения могут получить доступ к файлу
    private static final int PREFS_MODE = Context.MODE_PRIVATE;
    // для быстрого создания ключей можно использовать живой шаблон "key"
    private static final String KEY_BOOLEAN = "KEY_FOR_YOUR_BOOLEAN";
```

```

private static final String KEY_STRING = "KEY_FOR_YOUR_STRING";
private static final String KEY_FLOAT = "KEY_FOR_YOUR_FLOAT";
private static final String KEY_INT = "KEY_FOR_YOUR_INT";
private static final String KEY_LONG = "KEY_FOR_YOUR_LONG";

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

    // Получить сохраненный флаг (или значение по умолчанию, если он еще не был
    // сохранен)
    SharedPreferences settings = getSharedPreferences(PREFS_FILE, PREFS_MODE);
    // считывание булевого значения (по умолчанию false)
    boolean booleanVal = settings.getBoolean(KEY_BOOLEAN, false);
    // считывание значения int (по умолчанию 0)
    intVal = settings.getInt(KEY_INT, 0);
    // считывание строкового значения (по умолчанию "my string")
    String str = settings.getString(KEY_STRING, "my string");
    // считывание длинного значения (по умолчанию 123456)
    long longVal = settings.getLong(KEY_LONG, 123456);
    // считывание плавающего значения (по умолчанию 3.14f)
    float floatVal = settings.getFloat(KEY_FLOAT, 3.14f);
}

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

    // Сохранить флаг
    SharedPreferences settings = getSharedPreferences(PREFS_FILE, PREFS_MODE);
    SharedPreferences.Editor editor = settings.edit();
    // записать булево значение
    editor.putBoolean(KEY_BOOLEAN, true);
    // записать целочисленное значение
    editor.putInt(KEY_INT, 123);
    // записать строку
    editor.putString(KEY_STRING, "строковое значение");
    // записать значение long
    editor.putLong(KEY_LONG, 456876451);
    // записываем значение float
    editor.putFloat(KEY_FLOAT, 1.51f); editor.apply();
}
}
public String getPrefsString() {
    return prefStr;
}

```

`getSharedPreferences()` – это метод класса `Context`, который расширяет активность. Если необходимо получить доступ к методу `getSharedPreferences()` из других классов, то можно использовать `context.getSharedPreferences()` со ссылкой на объект `Context` из класса `Activity`, `View` или `Application`.

40.4. Извлечение всех сохраненных записей из определенного файла SharedPreferences

Метод `getAll()` извлекает все значения из предпочтений. Мы можем использовать его, например, для записи в журнал текущего содержимого `SharedPreferences`:

```

private static final String PREFS_FILE = "MyPrefs";

public static void logSharedPreferences(final Context context) {

```

```

SharedPreferences sharedPreferences = context.getSharedPreferences(PREFS_FILE,
    Context.MODE_PRIVATE);
Map<String, ?> allEntries = sharedPreferences.getAll();
for (Map.Entry<String, ?> entry : allEntries.entrySet()) {
    final String key = entry.getKey();
    final Object value = entry.getValue();
    Log.d("map values", key + ": " + value);
}
}
}

```

Документация предупреждает о модификации Collection, возвращаемой при помощи getAll:

Обратите внимание, что запрещается модифицировать коллекцию, возвращаемую этим методом, или изменять ее содержимое. Согласованность хранимых данных при этом не гарантируется.

40.5. Чтение и запись данных в SharedPreferences с помощью синглтон-класса

Рассмотрим синглтон-класс SharedPreferences Manager для чтения и записи всех типов данных:

```

import android.content.Context;
import android.content.SharedPreferences;
import android.util.Log;

import com.google.gson.Gson;
import java.lang.reflect.Type;

/**
 * Синглтон-класс для доступа к SharedPreferences,
 * должен быть инициализирован один раз в начале любым компонентом приложения,
 * использующим статический метод initialize(applicationContext)
 */
public class SharedPrefsManager {

    private static final String TAG = SharedPrefsManager.class.getName();
    private SharedPreferences prefs;
    private static SharedPrefsManager uniqueInstance;
    public static final String PREF_NAME = "com.example.app";

    private SharedPrefsManager(Context applicationContext) {
        prefs = applicationContext.getSharedPreferences(PREF_NAME, Context.MODE_PRIVATE);
    }

    /**
     * Выбрасывается IllegalStateException, если данный класс не инициализирован
     *
     * @return уникальный экземпляр SharedPrefsManager
     */
    public static SharedPrefsManager getInstance() {
        if (uniqueInstance == null) {
            throw new IllegalStateException(
                "SharedPrefsManager is not initialized, call
                initialize(applicationContext) " + "static method first");
        }
    }
}

```

```
    return uniqueInstance;
}

/**
 * Инициализируйте этот класс с помощью applicationContext,
 * должен вызываться один раз в начале работы каждым компонентом приложения
 *
 * @param applicationContext контекст приложения
 */
public static void initialize(Context applicationContext) {
    if (applicationContext == null) {
        throw new NullPointerException("Provided application context is null");
    }
    if (uniqueInstance == null) {
        synchronized (SharedPrefsManager.class) {
            if (uniqueInstance == null) {
                uniqueInstance = new SharedPrefsManager(applicationContext);
            }
        }
    }
    prefs = getPrefs();
    return prefs;
}

private SharedPreferences getPrefs() {
    return prefs;
}

/**
 * Очищает все данные в SharedPreferences
 */
public void clearPrefs() {
    SharedPreferences.Editor editor = getPrefs().edit();
    editor.clear();
    editor.commit();
}

public void removeKey(String key) {
    getPrefs().edit().remove(key).commit();
}

public boolean containsKey(String key) {
    return getPrefs().contains(key);
}

public String getString(String key, String defaultValue) {
    return getPrefs().getString(key, defaultValue);
}

public String getString(String key) {
    return getString(key, null);
}

public void setString(String key, String value) {
    SharedPreferences.Editor editor = getPrefs().edit();
    editor.putString(key, value);
    editor.apply();
}

public int getInt(String key, int defaultValue) {
    return getPrefs().getInt(key, defaultValue);
}
```

```

public int getInt(String key) {
    return getInt(key, 0);
}

public void setInt(String key, int value) {
    SharedPreferences.Editor editor = getPrefs().edit();
    editor.putInt(key, value);
    editor.apply();
}

public long getLong(String key, long defValue) {
    return getPrefs().getLong(key, defValue);
}

public long getLong(String key) {
    return getLong(key, 0L);
}

public void setLong(String key, long value) {
    SharedPreferences.Editor editor = getPrefs().edit();
    editor.putLong(key, value);
    editor.apply();
}

public boolean getBoolean(String key, boolean defValue) {
    return getPrefs().getBoolean(key, defValue);
}

public boolean getBoolean(String key) {
    return getBoolean(key, false);
}

public void setBoolean(String key, boolean value) {
    SharedPreferences.Editor editor = getPrefs().edit();
    editor.putBoolean(key, value);
    editor.apply();
}

public float getFloat(String key) {
    return getFloat(key, 0f);
}

public float getFloat(String key, float defValue) {
    return getFloat(key, defValue);
}

public void setFloat(String key, Float value) {
    SharedPreferences.Editor editor = getPrefs().edit();
    editor.putFloat(key, value);
    editor.apply();
}

/**
 * Сохраняет объект в prefs по указанному ключу, класс данного объекта должен
 * реализовывать интерфейс Model
 *
 * @param key             Стока
 * @param modelObject     Объект для сохранения
 * @param <M>              Generic для объекта

```

```

*/
public <M extends Model> void setObject(String key, M modelObject) {
    String value = createJSONStringFromObject(modelObject);
    SharedPreferences.Editor editor = getPrefs().edit();
    editor.putString(key, value);
    editor.apply();
}

/**
* Извлекает ранее сохраненный объект заданного класса из prefs
*
* @param key      Страна
* @param classOfModelObject Класс сохраняемого объекта
* @param <M>          Generic для объекта
* @return         Объект данного класса
*/
public <M extends Model> M getObject(String key, Class<M> classOfModelObject) {
    String jsonData = getPrefs().getString(key, null);
    if (null != jsonData) {
        try {
            Gson gson = new Gson();
            M customObject = gson.fromJson(jsonData, classOfModelObject);
            return customObject;
        } catch (ClassCastException cce) {
            Log.d(TAG, "Невозможно преобразовать строку, полученную из prefs, в коллекцию типа " + classOfModelObject.getName() + "\n" + cce.getMessage());
        }
    }
    return null;
}

/**
* Сохраняет объект Collection в prefs по указанному ключу
*
* @param keyСтрана
* @param dataCollection Объект коллекции
* @param <C>Generic для объекта Collection
*/
public <C> void setCollection(String key, C dataCollection) {
    SharedPreferences.Editor editor = getPrefs().edit();
    String value = createJSONStringFromObject(dataCollection);
    editor.putString(key, value);
    editor.apply();
}

/**
* Извлекает ранее сохраненный объект Collection Object заданного типа из prefs
*
* @param key      Страна
* @param typeOfC Тип объекта коллекции
* @param <C>          Generic для объекта коллекции
* @return         Объект коллекции, который может быть приведен
*/
public <C> C getCollection(String key, Type typeOfC) {
    String jsonData = getPrefs().getString(key, null);
    if (null != jsonData) {
        try {
            Gson gson = new Gson();

```

```

    C arrFromPrefs = gson.fromJson(jsonData, typeOfC);
    return arrFromPrefs;
} catch (ClassCastException cce) {
    Log.d(TAG, "Невозможно преобразовать строку, полученную из prefs, в коллекцию типа " + typeOfC.toString() + "\n" + cce.getMessage());
}
}
return null;
}

public void registerPrefsListener(SharedPreferences.OnSharedPreferenceChangeListener listener) {
    getPrefs().registerOnSharedPreferenceChangeListener(listener);
}

public void unregisterPrefsListener(
    SharedPreferences.OnSharedPreferenceChangeListener listener) {
    getPrefs().unregisterOnSharedPreferenceChangeListener(listener);
}

public SharedPreferences.Editor getEditor() {
    return getPrefs().edit();
}

private static String createJSONStringFromObject(Object object) {
    Gson gson = new Gson();
    return gson.toJson(object);
}
}

interface модели, который реализуется классами, обращающимися к Gson, чтобы избежать обfuscации proguard.

public interface Model {
}
}

Правила proguard для интерфейса Model:
-keep interface com.example.app.Model
-keep class * implements com.example.app.Model { *; }

```

40.6. getPreferences(int) и getSharedPreferences(String, int)

getPreferences(int)

возвращает предпочтения, сохраненные по имени класса активности, как описано в документации:

Получение объекта SharedPreferences для доступа к предпочтениям, которые являются частными для данной активности. Для этого достаточно вызвать базовый метод `getSharedPreferences(String, int)`, передав в качестве имени предпочтений имя класса данной активности.

При использовании метода `getSharedPreferences (String name, int mode)` возвращаются предпочтения, сохраненные под заданным именем name. Документация сообщает:

Получает и хранит содержимое файла предпочтений 'name', возвращая SharedPreferences, через которые можно получать и изменять его значения.

Поэтому, если значение, сохраняемое в SharedPreferences, должно использоваться во всем приложении, следует использовать `getSharedPreferences(String name, int mode)` с фиксированным именем, так как использование `getPreferences(int)` возвращает или сохраняет настройки, принадлежащие вызывающей его активности.

40.7. Прослушивание изменений SharedPreferences

```
SharedPreferences sharedPreferences = ...;
sharedPreferences.
registerOnSharedPreferenceChangeListener(mOnSharedPreferenceChangeListener);
```

```
private final SharedPreferences.OnSharedPreferenceChangeListener
mOnSharedPreferenceChangeListener = new SharedPreferences.
OnSharedPreferenceChangeListener() {
    @Override
    public void onSharedPreferenceChanged(SharedPreferences sharedPreferences,
    String key) {
        //TODO
    }
}
```

Обратите внимание!

- Слушатель сработает только в том случае, если значение было добавлено или изменено, установка того же значения не вызовет его.
- Слушатель должен быть сохранен в переменной-члене, а НЕ в анонимном классе, так как `registerOnSharedPreferenceChangeListener` хранит его в виде слабой ссылки, поэтому он будет собран в мусор (garbage collected).
- Вместо того чтобы использовать переменную-член, ее можно также реализовать непосредственно в классе и затем вызвать при помощи `registerOnSharedPreferenceChangeListener(this)`.
- Не забудьте отменить регистрацию слушателя, если он больше не нужен, используя `unregisterOnSharedPreferenceChangeListener`.

40.8. Сохранение, извлечение, удаление и очистка данных из SharedPreferences

Создание SharedPreferences `BuyyaPref`:

```
SharedPreferences pref = getApplicationContext().getSharedPreferences("BuyyaPref",
MODE_PRIVATE); Editor editor = pref.edit();
```

Хранение данных в виде пары «ключ-значение»:

```
editor.putBoolean("key_name1", true); // Сохранение булевых значений - true/false
editor.putInt("key_name2", 10); // Сохранение целочисленного значения
editor.putFloat("key_name3", 10.1f); // Сохранение значения типа float
editor.putLong("key_name4", 1000); // Сохранение значения типа long
editor.putString("key_name5", "MyString"); // Сохранение значения типа string

// Сохранить изменения в SharedPreferences
editor.commit(); // зафиксировать изменения
```

Получение данных SharedPreferences. Если значение для ключа не существует, то возвращается второе значение параметра (в данном случае null, то есть значение по умолчанию):

```
pref.getBoolean("key_name1", null); // получение булевых значений
pref.getInt("key_name2", null); // получение целочисленного значения
pref.getFloat("key_name3", null); // получение значения типа float
pref.getLong("key_name4", null); // получение значения типа long
pref.getString("key_name5", null); // получение значения типа string
```

Удаление значения ключа из SharedPreferences:

```
editor.remove("key_name3"); // будет удален ключ key_name3
editor.remove("key_name4"); // будет удален ключ key_name4

// Сохранить изменения в SharedPreferences
editor.commit(); // зафиксировать изменения
```

Очистить все данные из SharedPreferences:

```
editor.clear();
editor.commit(); // зафиксировать изменения
```

40.9. Добавление фильтра для EditTextPreference

Создадим класс:

```
public class InputFilterMinMax implements InputFilter {
    private int min, max;

    public InputFilterMinMax(int min, int max) {
        this.min = min;
        this.max = max;
    }

    public InputFilterMinMax(String min, String max) {
        this.min = Integer.parseInt(min);
        this.max = Integer.parseInt(max);
    }

    @Override
    public CharSequence filter(CharSequence source, int start, int end, Spanned dest, int dstart, int dend) {
        try {
            int input = Integer.parseInt(dest.toString() + source.toString());
            if (isInRange(min, max, input))
                return null;
        } catch (NumberFormatException nfe) { }
        return "";
    }

    private boolean isInRange(int a, int b, int c) {
        return b > a ? c >= a && c <= b : c >= b && c <= a;
    }
}
```

Используем его:

```
EditText compressPic = ((EditTextPreference) findPreference("pref_key_compress_pic")).getEditText();
compressPic.setFilters(new InputFilter[]{ new InputFilterMinMax(1, 100) });
```

40.10. Поддерживаемые типы данных в SharedPreferences

SharedPreferences позволяют хранить только примитивные типы данных (boolean, float, long, int, String и string set). Более сложные объекты хранить нельзя, по той причине, что как таковые SharedPreferences предназначены для хранения пользовательских настроек и т.п., это не база данных для хранения пользовательских данных (например, для сохранения списка дел, составленного пользователем).

Чтобы сохранить что-либо в SharedPreferences, используются ключи (Key) и значения (Value). Ключ – это то, как вы можете ссылаться на то, что вы сохранили позже, а значение – данные, которые вы хотите сохранить:

```
String keyToUseToFindLater = "High Score";
int newHighScore = 12938;
//получение объектов SharedPreferences и Editor
SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
//сохранение целочисленного значения в файле SharedPreferences
editor.putInt(keyToUseToFindLater, newHighScore);
editor.commit();
```

40.11. Различные способы инстанцирования объекта SharedPreferences

Получить доступ к SharedPreferences можно несколькими способами.

Получить файл SharedPreferences по умолчанию:

```
import android.preference.PreferenceManager;
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
```

Получить конкретный файл SharedPreferences:

```
public static final String PREF_FILE_NAME = "PrefFile";
SharedPreferences prefs = getSharedPreferences(PREF_FILE_NAME, MODE_PRIVATE);
```

Получить SharedPreferences из другого приложения:

```
// Обратите внимание, что другое приложение должно объявить предпочтения prefs как
MODE_WORLD_WRITEABLE
final ArrayList<HashMap<String, String>> LIST = new ArrayList<HashMap<String, String>>();
Context contextOtherApp = createPackageContext("com.otherapp", Context.MODE_WORLD_
WRITEABLE);
SharedPreferences prefs = contextOtherApp.getSharedPreferences("pref_file_name",
Context.MODE_WORLD_READABLE);
```

40.12. Удаление ключей

```
private static final String MY_PREF = "MyPref";
// ...
```

```

SharedPreferences prefs = ...;

// ...

SharedPreferences.Editor editor = prefs.edit();
editor.putString(MY_PREF, "value");
editor.remove(MY_PREF);
editor.apply();

```

После выполнения `apply()` предпочтения `prefs` содержит пары “key” -> “value” в дополнение к тому, что уже содержалось. Несмотря на то, что выглядит так, как будто сначала “key” добавляется, а затем удаляется, на самом деле сначала происходит удаление. Изменения в редакторе применяются все сразу, а не в том порядке, в котором вы их добавляли. Все удаления происходят раньше всех добавлений.

Глава 41. Намерение (Intent)

Параметр	Подробности
<code>intent</code>	Начало намерения
<code>requestCode</code>	Уникальный номер для идентификации запроса
<code>options</code>	Дополнительные опции запуска активности
<code>name</code>	Имя дополнительных данных
<code>value</code>	Значение дополнительных данных
<code>CHOOSE_CONTACT_REQUEST_CODE</code>	Код запроса для его идентификации в методе <code>onActivityResult</code>
<code>action</code>	Любое действие, которое необходимо выполнить с помощью данного намерения, например <code>Intent.ACTION_VIEW</code>
<code>uri</code>	<code>uri</code> данных, который будет использоваться намерением для выполнения указанного действия
<code>packageContext</code>	Контекст, используемый для инициализации намерения
<code>cls</code>	Класс, который будет использоваться данным намерением

Намерение (Intent) – это небольшое сообщение, передаваемое по системе Android. Это сообщение может содержать информацию о нашем намерении выполнить ту или иную задачу.

По сути, это пассивная структура данных, содержащая абстрактное описание выполняемого действия.

41.1. Получение результата от другой активности

С помощью метода `startActivityForResult(Intent intent, int requestCode)` можно запустить другую активность, а затем получить от нее результат в методе `onActivityResult(int requestCode, int resultCode, Intent data)`. Результат будет возвращен в виде намерения (Intent). Намерение может передавать данные при помощи пакета (Bundle).

В данном примере `MainActivity` будет запускать `DetailActivity` и затем ожидать от нее результата. Каждый тип запроса должен иметь свой код запроса (`int`), чтобы в *определенном* методе `onActivityResult(int requestCode, int resultCode, Intent data)`

В MainActivity можно было определить, какой запрос обрабатывать, сравнив значения requestCode и REQUEST_CODE_EXAMPLE (хотя в данном примере он только один).

MainActivity:

```

public class MainActivity extends Activity {

    // Использование уникального кода запроса для каждого случая использования
    private static final int REQUEST_CODE_EXAMPLE = 0x9345;

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

        // Создание нового экземпляра Intent для запуска DetailActivity
        final Intent intent = new Intent(this, DetailActivity.class);

        // Запуск DetailActivity с кодом запроса
        startActivityForResult(intent, REQUEST_CODE_EXAMPLE);
    }

    // вызывается только onActivityResult
    // когда другая активность ранее была запущена с помощью startActivityForResult
    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        // Сначала нужно проверить, совпадает ли requestCode с тем, который мы
        // использовали.
        if(requestCode == REQUEST_CODE_EXAMPLE) {

            // Код результата задается действием DetailActivity
            // По условию RESULT_OK означает, что все, что
            // совершило DetailActivity, было выполнено успешно
            if(resultCode == Activity.RESULT_OK) {
                // Получить результат из возвращенного намерения
                final String result = data.getStringExtra(DetailActivity.EXTRA_DATA);

                // Использовать данные - в этом случае вывести на экран в виде тоста
                Toast.makeText(this, "Result: " + result, Toast.LENGTH_LONG).show();
            } else {
                // setResult не был успешно выполнен DetailActivity
                // из-за какой-то ошибки или потока управления. Нет данных для извлечения
            }
        }
    }
}

```

DetailActivity:

```

public class DetailActivity extends Activity {

    // Константа для идентификации данных, передаваемых между активностями.
    public static final String EXTRA_DATA = "EXTRA_DATA";

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

```

```

final Button button = (Button) findViewById(R.id.button);
// При клике на эту кнопку мы хотим вернуть результат
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Создаем новый объект Intent в качестве контейнера для результата
        final Intent data = new Intent();
        // Добавьте необходимые данные для возврата в MainActivity
        data.putExtra(EXTRA_DATA, "Некоторые интересные данные!");

        // Установить код результата Activity.RESULT_OK,
        // чтобы указать успех и присоединить намерение,
        // которое содержит наши результирующие данные
        setResult(Activity.RESULT_OK, data);

        // С помощью функции finish() мы закрываем DetailActivity для
        // возврата в MainActivity
        finish();
    }
});
```

@Override

```

public void onBackPressed() {
    // При нажатии пользователем кнопки "Назад" установить resultCode
    // как Activity.RESULT_CANCELED для индикации неудачи
    setResult(Activity.RESULT_CANCELED);
    super.onBackPressed();
}
```

}

Несколько моментов, о которых необходимо знать

- Данные возвращаются только после вызова функции `finish()`. Перед вызовом функции `finish()` необходимо вызвать функцию `setResult()`, иначе результат возвращен не будет.
- Убедитесь, что в `Activity` не используется `android:launchMode="singleTask"`, иначе активность будет выполняться в отдельной задаче, и, соответственно, вы не получите от нее результата. Если `Activity` использует режим запуска `singleTask`, то она сразу же вызовет `onActivityResult()` с кодом результата `Activity.RESULT_CANCELED`.
- При вызове функции `startActivityForResult()` можно использовать явные или неявные намерения. При запуске одной из собственных активностей для получения результата следует использовать явное намерение, чтобы гарантированно получить ожидаемый результат. Явное намерение всегда доставляется к его цели, независимо от того, что оно содержит; фильтр при этом не используется. Неявное же намерение доставляется компоненту только в том случае, если оно может пройти через один из фильтров компонента.

41.2. Передача данных между активностями

Данный пример иллюстрирует отправку строки `String` со значением "Some data!" из `OriginActivity` в `DestinationActivity`.

Примечание: это наиболее простой способ передачи данных между двумя действиями. Более надежная реализация приведена в примере использования паттерна `starter`.

OriginActivity:

```

public class OriginActivity extends AppCompatActivity {

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

        // Создаем новый объект намерения, содержащий DestinationActivity в качестве
        // целевой активности.
        final Intent intent = new Intent(this, DestinationActivity.class);

        // Добавьте данные в виде пар "ключ-значение" в объект намерения с помощью
        // функции putExtra()
        intent.putExtra(DestinationActivity.EXTRA_DATA, "Some data!");

        // Запуск целевой активности с объектом намерения
        startActivity(intent);
    }
}

```

DestinationActivity:

```

public class DestinationActivity extends AppCompatActivity {

    public static final String EXTRA_DATA = "EXTRA_DATA";

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

        // getIntent() возвращает объект намерения, который был использован
        // для запуска данной активности
        final Intent intent = getIntent();
        // Получение данных из объекта намерения с использованием того же ключа,
        // что ранее использовался для добавления данных в объект намерения
        // в OriginActivity.
        final String data = intent.getStringExtra(EXTRA_DATA);
    }
}

```

Можно передавать и другие примитивные типы данных, массивы, данные Bundle и Parcelable. Передача Serializable также возможна, но ее следует избегать, так как она более чем в три раза медленнее, чем Parcelable.

Serializable – это стандартный интерфейс Java. Вы просто помечаете класс как Serializable, реализуя интерфейс Serializable, и Java будет автоматически сериализовать его в необходимых ситуациях.

Parcelable – это специфический для Android интерфейс, который может быть реализован на пользовательских типах данных (то есть на ваших собственных объектах / POJO-объектах), это позволяет вашему объекту быть «уплощенным» (flattened) и реконструировать себя без необходимости делать что-либо в месте назначения.

Когда у вас есть объект parcelable, вы можете отправить его, как примитивный тип, с объектом намерения:

```
intent.putExtra(DestinationActivity.EXTRA_DATA, myParcelableObject);
```

Или в связке / в качестве аргумента для фрагмента:

```
bundle.putParcelable(DestinationActivity.EXTRA_DATA, myParcelableObject);
```

...а затем также считать этот объект из намерения в пункте назначения с помощью `getParcelableExtra`:

```
final MyParcelableType data = intent.getParcelableExtra(EXTRA_DATA);
```

Или при чтении фрагмента из бандла (пакета):

```
final MyParcelableType data = bundle.getParcelable(EXTRA_DATA);
```

Получив объект `Serializable`, вы можете поместить его в объект намерения:

```
bundle.putSerializable(DestinationActivity.EXTRA_DATA, mySerializableObject);
```

...а затем также считать его из объекта намерения в месте назначения, как показано ниже:

```
final SerializableType data = (SerializableType)bundle.getSerializable(EXTRA_DATA);
```

41.3. Открытие URL-адреса в браузере

Открытие в браузере по умолчанию

В этом примере показано, как можно программно открыть URL-адрес во встроенным веб-браузере, а не внутри приложения. Это позволяет приложению открывать веб-страницу без необходимости включать разрешение `INTERNET` в файл манифеста.

```
public void onBrowseClick(View v) {
    String url = "http://www.google.com";
    Uri uri = Uri.parse(url);
    Intent intent = new Intent(Intent.ACTION_VIEW, uri);
    // Проверить, что намерение будет разрешено в активность
    if (intent.resolveActivity(getApplicationContext()) != null) {
        // Здесь мы используем намерение без Chooser, в отличие от следующего примера
        startActivityForResult(intent);
    }
}
```

Предложение пользователю выбрать браузер

Обратите внимание, что в данном примере используется метод `Intent.createChooser()`:

```
public void onBrowseClick(View v) {
    String url = "http://www.google.com";
    Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
    // Обратите внимание на расположенный ниже Chooser. Если ни одно приложение
    // не подходит, Android выводит системное сообщение. Поэтому здесь нет
    // необходимости в try-catch.
    startActivityForResult(Intent.createChooser(intent, "Browse with"));
}
```

В некоторых случаях URL может начинаться с “`www`”. В этом случае вы получите данное исключение:

`android.content.ActivityNotFoundException`: Не найдено ни одной активности для обработки намерения

URL всегда должен начинаться с "`http://`" или "`https://`". Поэтому ваш код должен проверять его наличие, как показано в следующем фрагменте:

```
if (!url.startsWith("https://") && !url.startsWith("http://")){
    url = "http://" + url;
}
Intent openUrlIntent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
if (openUrlIntent.resolveActivity(getApplicationContext()) != null) {
    startActivity(openUrlIntent);
}
```

Лучшие практики

Проверьте наличие на устройстве приложений, которые могут получить неявное намерение, чтобы при вызове функции `startActivity()` не произошел сбой. Чтобы убедиться в существовании приложения, способного получить намерение, вызовите `resolveActivity()` для объекта Intent. Если результат не равен `null`, то существует по крайней мере одно приложение, которое может обработать намерение, и можно вызывать `startActivity()`. Если результат равен `null`, то намерение не следует использовать и, если возможно, следует отключить функцию, вызывающую намерение.

41.4. Паттерн стартера

Этот паттерн представляет собой более строгий подход к запуску активности. Его цель – улучшить читаемость кода и в то же время снизить его сложность, затраты на сопровождение и связь компонентов.

Следующий пример реализует паттерн starter, который обычно реализуется в виде статического метода на самой активности. Этот статический метод принимает все необходимые параметры, составляет из этих данных корректное намерение и запускает активность.

Как известно, намерение – это объект, обеспечивающий связь между отдельными компонентами, например двумя действиями, во время выполнения программы. Он представляет собой “намерение (intent) приложения сделать что-то”. Намерения можно использовать для решения самых разных задач, но в данном случае намерение запускает другую активность.

```
public class ExampleActivity extends AppCompatActivity {

    private static final String EXTRA_DATA = "EXTRA_DATA";

    public static void start(Context context, String data) {
        Intent intent = new Intent(context, ExampleActivity.class);
        intent.putExtra(EXTRA_DATA, data);
        context.startActivity(intent);
    }

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

        Intent intent = getIntent();
        if(!intent.getExtras().containsKey(EXTRA_DATA)){
            throw new UnsupportedOperationException("Activity should be started using the static start method");
        }
        String data = intent.getStringExtra(EXTRA_DATA);
    }
}
```

Этот паттерн также позволяет принудительно передавать дополнительные данные вместе с намерением. Запуск ExampleActivity может быть осуществлен следующим образом, где context – контекст активности:

```
ExampleActivity.start(context, "Некоторые данные!");
```

41.5. Очистка стека активности

Иногда требуется начать новое действие, удалив при этом предыдущие действия из обратного стека, чтобы кнопка “Назад” не возвращала вас к ним. Примером может быть запуск приложения на активности Login, которое приводит вас к активности Main вашего приложения, но при выходе из приложения вы хотите, чтобы вас направляли обратно на Login без возможности вернуться назад. В таком случае можно установить для намерения флаг FLAG_ACTIVITY_CLEAR_TOP, то есть если запускаемое действие уже выполняется в текущей задаче (LoginActivity), то вместо запуска нового экземпляра этого действия все другие действия, расположенные поверх него, будут закрыты, а данное намерение будет доставлено старому (теперь уже верхнему) действию как новое намерение.

```
Intent intent = new Intent(getApplicationContext(), LoginActivity.class);
intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intent);
```

Также можно использовать флаги FLAG_ACTIVITY_NEW_TASK вместе с FLAG_ACTIVITY_CLEAR_TASK, если необходимо очистить все активности на обратном стеке:

```
Intent intent = new Intent(getApplicationContext(), LoginActivity.class);
// Закрытие всех активностей, очистка обратного стека.
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
startActivity(intent);
```

41.6. Запуск активности

В этом примере DestinationActivity будет запускаться из OriginActivity. Здесь Intent-конструктор принимает два параметра:

1. Context в качестве первого параметра (используется потому, что класс Activity является подклассом Context)
2. Class компонента приложения, которому система должна доставить намерение (в данном случае это активность, которая должна быть запущена)

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

        Intent intent = new Intent(this, DestinationActivity.class);

        startActivity(intent);
        finish(); // Опционально можно закрыть OriginActivity. Таким образом, когда
        // пользователь нажмет кнопку возврата из DestinationActivity, то больше не попадет
        // в OriginActivity.
    }
}
```

Другой способ создания намерения открыть `DestinationActivity` – использовать конструктор по умолчанию для намерений и с помощью метода `setClass()` указать ему, какую `Activity` следует открыть:

```
Intent i=new Intent();
setClass(this, DestinationActivity.class);
startActivity(intent);
finish(); // При желании можно закрыть OriginActivity. Таким образом, при нажатии "назад" из DestinationActivity пользователь не попадет в OriginActivity
```

41.7. Отправка электронных писем

```
// Составим Uri со схемой 'mailto'
Intent emailIntent = new Intent(Intent.ACTION_SENDTO, Uri.fromParts("mailto",
"johndoe@example.com", null));
// Тема
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Hello World!");
// Тело письма
emailIntent.putExtra(Intent.EXTRA_TEXT, "Hi! I am sending you a test email.");
// Приложенный файл
emailIntent.putExtra(Intent.EXTRA_STREAM, attachedFileUri);

// Проверим, есть ли на устройстве почтовый клиент
if (emailIntent.resolveActivity(getApplicationContext()) != null) {
    // Предложим пользователю выбрать почтовое приложение
    startActivityForResult(Intent.createChooser(emailIntent, "Choose your mail
application"));
} else {
    // Проинформируем пользователя об отсутствии установленных почтовых клиентов
    // или предоставим альтернативу
}
```

Этот код позволит предварительно заполнить письмо в выбранном пользователем почтовом приложении.

Если необходимо добавить вложение, то можно использовать `Intent.ACTION_SEND` вместо `Intent.ACTION_SENDTO`. Для нескольких вложений можно использовать `ACTION_SEND_MULTIPLE`.

Предупреждение: не каждое устройство имеет провайдера для `ACTION_SENDTO`, и вызов `startActivity()` без предварительной проверки с помощью `resolveActivity()` может вызвать исключение `ActivityNotFoundException`.

41.8. CustomTabsIntent для пользовательских вкладок Chrome

С помощью `CustomTabsIntent` можно настраивать пользовательские вкладки Chrome (см. <https://developer.chrome.com/multidevice/android/customtabs>) для настройки ключевых компонентов пользовательского интерфейса в браузере, открываемом из вашего приложения.

Для некоторых случаев это хорошая альтернатива использованию `WebView`, позволяющая загружать веб-страницу с помощью намерения, при этом появляется возможность в некоторой степени привнести в браузер внешний вид и функциональность вашего приложения.

Приведем пример открытия url с помощью `CustomTabsIntent`:

```
String url = "https://www.google.pl/";
CustomTabsIntent intent = new CustomTabsIntent.Builder()
    .setStartAnimations(getApplicationContext(), R.anim.slide_in_right, R.anim.slide_out_left)
```

```

.setExitAnimations(getContext(), android.R.anim.slide_in_left,
    android.R.anim.slide_out_right)
.setCloseButtonIcon(BitmapFactory.decodeResource(getResources(), R.drawable.
    ic_arrow_back_white_24dp))
.setToolbarColor(Color.parseColor("#43A047"))
.enableUrlBarHiding()
.build();
intent.launchUrl(getActivity(), Uri.parse(url));

```

Примечание: чтобы использовать пользовательские вкладки, необходимо добавить соответствующую зависимость в файл build.gradle:

```
compile 'com.android.support:customtabs:24.1.1'
```

41.9. Намерение URI

В данном примере показано, как запустить намерение из браузера:

```
<a href="intent://host.com/path#Intent;package=com.sample.
test;scheme=yourscheme;end">Start intent</a>
```

Это намерение запустит приложение с пакетом com.sample.test или откроет Google Play с этим пакетом. Также это намерение может быть запущено с помощью:

```
var intent = "intent://host.com/path#Intent;package=com.sample.
test;scheme=yourscheme;end"; window.location.replace(intent)
```

В активности этот хост и путь могут быть получены из данных намерения:

```

@Override
public void onCreate(Bundle bundle) {
    super.onCreate(bundle);
    Uri data = getIntent().getData();
    // возвращает host.com/path
}

```

Синтаксис URI намерения:

```

HOST/URI-path // Необязательный хост
#Intent;
    package=[string];
    action=[string];
    category=[string];
    component=[string];
    scheme=[string];
end;

```

41.10. Запуск звонка

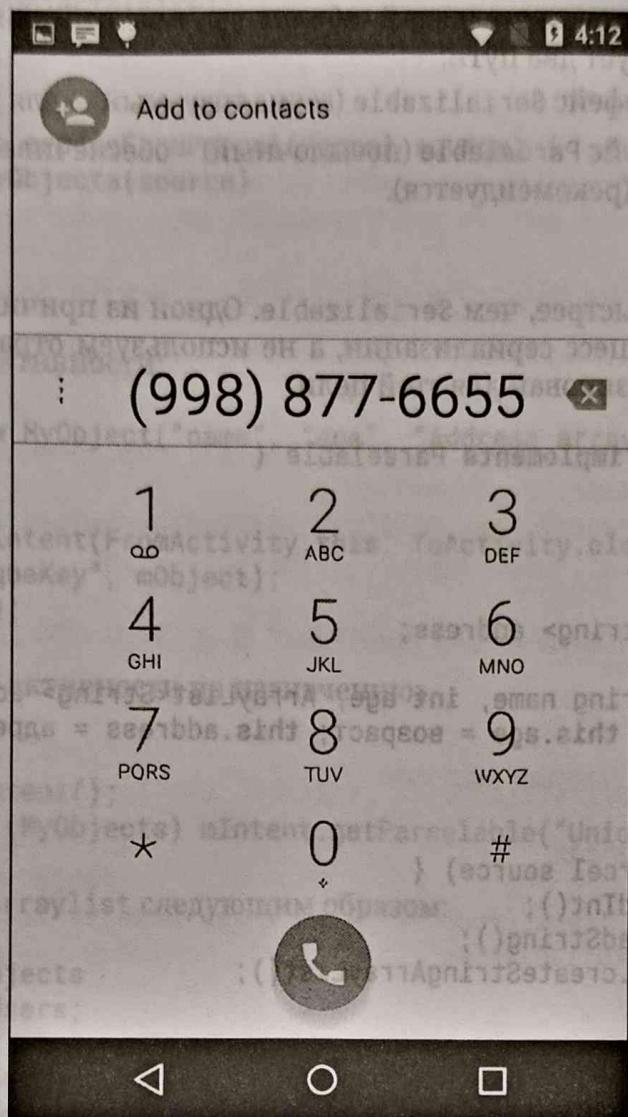
В данном примере показано, как открыть стандартный дозвонщик (dialer; приложение, осуществляющее обычные звонки) с уже заданным телефонным номером:

```

Intent intent = new Intent(Intent.ACTION_DIAL);
intent.setData(Uri.parse("tel:9988776655"));
// Замените номер телефона на действительный. Не забудьте добавить префикс tel|,
иначе произойдет сбой.
startActivity(intent);

```

Результат выполнения приведенного выше кода:



41.11. Передача широковещательных сообщений другим компонентам

Намерения могут использоваться для передачи широковещательных сообщений другим компонентам приложения (например, работающей фоновой службе) или всей системе Android.

Для отправки широковещательной рассылки внутри вашего приложения используйте класс LocalBroadcastManager:

```
Intent intent = new Intent("com.example.YOUR_ACTION"); // действие намерения
intent.putExtra("key", "value"); // данные, которые будут переданы с вашей трансляцией
```

```
LocalBroadcastManager manager = LocalBroadcastManager.getInstance(context);
manager.sendBroadcast(intent);
```

Для отправки широковещательной рассылки (трансляции) компонентам, находящимся **вне приложения**, используйте метод sendBroadcast() для объекта контекста Context:

```
Intent intent = new Intent("com.example.YOUR_ACTION"); // действие намерения
intent.putExtra("key", "value"); // данные, которые будут переданы с вашей трансляцией
context.sendBroadcast(intent);
```

41.12. Передача пользовательского объекта между активностями

Можно передавать пользовательский объект другим действиям с помощью класса `Bundle`. Для этого существует два пути:

- использовать интерфейс `Serializable` (сериализуемый) – для Java и Android;
- применить интерфейс `Parcelable` (посылочный) – обеспечивает экономию памяти, только для Android (рекомендуется).

Parcelable

Этот способ гораздо быстрее, чем `Serializable`. Одной из причин этого является то, что мы явно используем процесс сериализации, а не используем отражение. Кроме того, код был значительно оптимизирован для этой цели.

```
public class MyObjects implements Parcelable {  
  
    private int age;  
    private String name;  
  
    private ArrayList<String> address;  
  
    public MyObjects(String name, int age, ArrayList<String> address) {  
        this.name = имя; this.age = возраст; this.address = адрес;  
    }  
  
    public MyObjects(Parcel source) {  
        age = source.readInt();  
        name = source.readString();  
        address = source.createStringArrayList();  
    }  
  
    @Override  
    public int describeContents() {  
        return 0;  
    }  
  
    @Override  
    public void writeToParcel(Parcel dest, int flags) {  
        dest.writeInt(age);  
        dest.writeString(name);  
        dest.writeStringList(address);  
    }  
  
    public int getAge() {  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public ArrayList<String> getAddress() {  
        if (!address == null)  
            return address;  
        else  
            return new ArrayList<String>();  
    }  
  
    public static final Creator<MyObjects> CREATOR = new Creator<MyObjects>() {  
        @Override
```

```

public MyObjects[] newArray(int size) {
    return new MyObjects[size];
}

@Override
public MyObjects createFromParcel(Parcel source) {
    return new MyObjects(source);
}
};

}

```

Код отправляющей активности:

```
MyObject mObject = new MyObject("name", "age", "Address array here");
```

```
//Передача MyObject
```

```
Intent mIntent = new Intent(FromActivity.this, ToActivity.class);
mIntent.putExtra("UniqueKey", mObject);
startActivity(mIntent);
```

Получение объекта в активности по назначению:

```
//Получение MyObjects
```

```
Intent mIntent = getIntent();
MyObjects workorder = (MyObjects) mIntent.getParcelable("UniqueKey");
```

Возможно передать ArrayList следующим образом:

```
//Массив объектов MyObjects
ArrayList<MyObject> mUsers;
```

```
//Передача списка объектов MyObject
```

```
Intent mIntent = new Intent(FromActivity.this, ToActivity.class);
mIntent.putParcelableArrayListExtra("UniqueKey", mUsers);
startActivity(mIntent);
```

```
//Получение списка объектов MyObject
```

```
Intent mIntent = getIntent();
ArrayList<MyObjects> mUsers = mIntent.getParcelableArrayList("UniqueKey");
```

Примечание: для генерации Parcelable-кода существуют плагины для Android Studio, например:
<https://github.com/mcharmas/android-parcelable-intellij-plugin>.

Serializable

Код отправляющей активности:

```
Product product = new Product();
Bundle bundle = new Bundle();
bundle.putSerializable("product", product);
Intent cartIntent = new Intent(mContext,
ShowCartActivity.class); cartIntent.putExtras(bundle); mContext.
startActivity(cartIntent);
```

Получение объекта в целевой активности:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```

Bundle bundle = this.getIntent().getExtras();
Product product = null;
if (bundle != null) {
    product = (Product) bundle.getSerializable("product");
}

```

ArrayList Serializable-объектов передается так же, как и одиночный объект. Пользовательский объект должен реализовывать интерфейс Serializable.

41.13. Передача широты и долготы на Google Maps

Вы можете передавать широту и долготу из своего приложения на Google Maps с помощью намерения:

```

String uri = String.format(Locale.ENGLISH, "http://maps.google.com/
maps?q=loc:%f,%f", 28.43242324, 77.8977673);
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(uri));
startActivity(intent);

```

41.14. Передача различных данных в активность при помощи намерения

1. Передача целочисленных данных

Активность отправителя:

```

Intent myIntent = new Intent(SenderActivity.this, ReceiverActivity.class);
myIntent.putExtra("intVariableName", intValue);
startActivity(myIntent);

```

Активность получателя:

```

Intent mIntent = getIntent();
int intValue = mIntent.getIntExtra("intVariableName", 0); // установить 0 в качестве
значения по умолчанию, если значение для intVariableName не найдено

```

2. Передача данных типа double

Активность отправителя:

```

Intent myIntent = new Intent(SenderActivity.this, ReceiverActivity.class);
myIntent.putExtra("doubleVariableName", doubleValue);
startActivity(myIntent);

```

Активность получателя:

```

Intent mIntent = getIntent();
double doubleValue = mIntent.getDoubleExtra("doubleVariableName", 0.00);
// установить 0.00 в качестве значения по умолчанию, если значение
doubleVariableName не найдено

```

3. Передача строковых данных

Активность отправителя:

```

Intent myIntent = new Intent(SenderActivity.this, ReceiverActivity.class);
myIntent.putExtra("stringVariableName", stringValue);
startActivity(myIntent);

```

Активность получателя:

```
Intent mIntent = getIntent();
String stringValue = mIntent.getStringExtra("stringVariableName");
```

или

```
Intent mIntent = getIntent();
String stringValue = mIntent.getStringExtra("stringVariableName");
```

4. Передача данных ArrayList

Активность отправителя:

```
Intent myIntent = new Intent(SenderActivity.this, ReceiverActivity.class);
myIntent.putStringArrayListExtra("arrayListVariableName", arrayList);
startActivity(myIntent);
```

Активность получателя:

```
Intent mIntent = getIntent();
arrayList = mIntent.getStringArrayListExtra("arrayListVariableName");
```

5. Передача данных Object

Активность отправителя:

```
Intent myIntent = new Intent(SenderActivity.this, ReceiverActivity.class);
myIntent.putExtra("ObjectVariableName", yourObject);
startActivity(myIntent);
```

Активность получателя:

```
Intent mIntent = getIntent();
yourObj = mIntent.getSerializableExtra("ObjectVariableName");
```

Примечание: следует помнить, что ваш пользовательский класс должен реализовывать интерфейс Serializable.

6. Передача данных HashMap<String, String>

Активность отправителя:

```
HashMap<String, String> hashMap;
Intent mIntent = new Intent(SenderActivity.this, ReceiverActivity.class);
mIntent.putExtra("hashMap", hashMap);
startActivity(mIntent);
```

Активность получателя:

```
Intent mIntent = getIntent();
HashMap<String, String> hashMap = (HashMap<String, String>)
mIntent.getSerializableExtra("hashMap");
```

7. Передача Bitmap-данных

Активность отправителя:

```
Intent myIntent = new Intent(SenderActivity.this, ReceiverActivity.class);
myIntent.putExtra("image", bitmap);
startActivity(mIntent);
```

Активность получателя:

```
Intent mIntent = getIntent();
Bitmap bitmap = mIntent.getParcelableExtra("image");
```

41.15. ShareIntent

Обмен простой информацией с различными приложениями:

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, "Это мой текст для отправки.");
sendIntent.setType("text/plain");
startActivity(Intent.createChooser(sendIntent, getResources().getText(R.string.send_to)));
```

Для обмена изображениями с различными приложениями используем shareIntent:

```
Intent shareIntent = new Intent();
shareIntent.setAction(Intent.ACTION_SEND);
shareIntent.putExtra(Intent.EXTRA_STREAM, uriToImage);
shareIntent.setType("image/jpeg");
startActivity(Intent.createChooser(shareIntent, getResources().getText(R.string.send_to)));
```

41.16. Отображение выбора файла и чтение результата

Запуск активности выбора файла

```
public void showFileChooser() {
    Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
    // Обновление mime-типов
    intent.setType("*/*");
    // Обновите здесь дополнительные mime-типы, используя String[].
    intent.putExtra(Intent.EXTRA_MIME_TYPES, mimeTypes);
    // Выбираем только открываемые и локальные файлы. Теоретически мы можем брать
    // файлы с Google Drive или других приложений, работающих с сетевыми файлами,
    // но для данного примера это излишне.
    intent.addCategory(Intent.CATEGORY_OPENABLE);
    intent.putExtra(Intent.EXTRA_LOCAL_ONLY, true);
    // REQUEST_CODE = <некоторое целочисленное значение>
    startActivityForResult(intent, REQUEST_CODE);
}
```

Чтение результатов

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
```

```

// Если пользователь не выбрал файл, просто возвращаем
if (requestCode != REQUEST_CODE || resultCode != RESULT_OK) {
    return;
}

// Импорт файла
importFile(data.getData());
}

public void importFile(Uri uri) {
    String fileName = getFileName(uri);

    // Временный файл может быть любым, каким вы хотите
    File fileCopy = copyToTempFile(uri, File tempFile)

    // Готово!
}

/**
 * Получение имени файла для URI с помощью разрешителей содержимого (content resolvers).
 * Взято из: https://developer.android.com/training/secure-file-sharing/retrieve-
info.html#RetrieveFileInfo
 */
* @param uri - uri для запроса
*/
private String getFileName(Uri uri) throws IllegalArgumentException {
    // Получение курсора с информацией о данном uri
    Cursor cursor = getContentResolver().query(uri, null, null, null, null);

    if (cursor.getCount() <= 0) {
        cursor.close();
        throw new IllegalArgumentException("Can't obtain file name, cursor is empty
        (Невозможно получить имя файла, курсор пуст)");
    }

    cursor.moveToFirst();

    String fileName = cursor.getString(cursor.getColumnIndexOrThrow(OpenableColumns.
DISPLAY_NAME));

    cursor.close();

    return fileName;
}

/**
 * Копирует ссылку uri во временный файл
 *
 * @param uri - uri, используемый в качестве входного потока
 * @param tempFile - файл, используемый в качестве выходного потока
 * @return - входной tempFile для удобства
 * @throws - IOException при ошибке
 */
private File copyToTempFile(Uri uri, File tempFile) throws IOException {
    // Получение входного потока из uri
    InputStream inputStream = getContentResolver().openInputStream(uri);

    if (inputStream == null) {
        throw new IOException("Unable to obtain input stream from URI (Невозможно
        получить входной поток из URI)");
    }
}

```

```
// Копирование потока во временный файл
FileUtils.copyInputStreamToFile(inputStream, tempFile);

return tempFile;
}
```

41.17. Совместное использование нескольких файлов с помощью намерения

Список строк, передаваемый в качестве параметра методу `share()`, содержит пути ко всем файлам, к которым вы хотите предоставить общий доступ. По сути, метод перебирает все пути, добавляет их в Uri и запускает активность, которая может принимать файлы этого типа.

```
public static void share(AppCompatActivity context, List<String> paths) {

    if (paths == null || paths.size() == 0) {
        return;
    }
    ArrayList<Uri> uris = new ArrayList<>();
    Intent intent = new Intent();
    intent.setAction(android.content.Intent.ACTION_SEND_MULTIPLE);
    intent.setType("/*");
    for (String path : paths) {
        File file = new File(path);
        uris.add(Uri.fromFile(file));
    }
    intent.putParcelableArrayListExtra(Intent.EXTRA_STREAM, uris);
    context.startActivity(intent);
}
```

41.18. Запуск несвязанной службы с помощью намерения

Служба (Service) – это компонент, работающий в фоновом режиме (на потоке пользовательского интерфейса) без непосредственного взаимодействия с пользователем. Несвязанная служба только запускается и не привязана к жизненному циклу какой-либо активности.

Для запуска службы можно поступить так, как показано в примере ниже:

```
// Это намерение будет использоваться для запуска службы
Intent i = new Intent(context, ServiceName.class);
// потенциально добавить данные в дополнительные функции намерения
extras i.putExtra("KEY1", "Значение, которое будет использоваться службой");
context.startService(i);
```

С помощью переопределения `onStartCommand()` можно использовать любые дополнения из намерения:

```
public class MyService extends Service {
    public MyService() {
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId)
    {
        if (intent != null) {
            Bundle extras = intent.getExtras();
            String key1 = extras.getString("KEY1", "");
            if (key1.equals("Значение, которое будет использоваться службой")) {
```

```

        // сделать что-нибудь
    }

    return START_STICKY;
}

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

```

41.19. Получение результата из активности во фрагмент

Для получения результата из другой активности необходимо вызвать метод фрагмента `startActivityForResult(Intent intent, int requestCode)`. Обратите внимание, что не следует вызывать `getActivity().startActivityForResult()`, так как в этом случае результат будет возвращен в родительскую активность фрагмента.

Получить результат можно с помощью метода фрагмента `onActivityResult()`. При этом необходимо убедиться, что родительская активность фрагмента также переопределяет `onActivityResult()` и вызывает суперреализацию.

В следующем примере `ActivityOne` содержит фрагмент `FragmentOne`, который запустит активность `ActivityTwo` и будет ожидать от нее результата.

ActivityOne

```

public class ActivityOne extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_one);
    }

    // Вы должны переопределить этот метод, так как вторая активность всегда будет
    // отправлять свои результаты в эту активность, а затем во фрагмент
    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
    }
}

```

activity_one.xml

```

<fragment android:name="com.example.FragmentOne"
          android:id="@+id/fragment_one"
          android:layout_width="match_parent"
          android:layout_height="match_parent" />

```

FragmentOne

```

public class FragmentOne extends Fragment {
    public static final int REQUEST_CODE = 11;
    public static final int RESULT_CODE = 12;
    public static final String EXTRA_KEY_TEST = "testKey";

    // Инициализация и запуск второй активности ActivityTwo
    private void startSecondActivity() {

```

```

Intent intent = new Intent(getActivity(), ActivityTwo.class);
startActivityForResult(REQUEST_CODE, intent);
}

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == REQUEST_CODE && resultCode == RESULT_CODE) {
        String testResult = data.getStringExtra(EXTRA_KEY_TEST);
        // TODO: Сделать что-нибудь с вашими дополнительными данными
    }
}
}

```

ActivityTwo

```

public class ActivityTwo extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_two);
    }

    private void closeActivity() {
        Intent intent = new Intent();
        intent.putExtra(FragmentOne.EXTRA_KEY_TEST, "Тестирование передачи данных
        обратно в ActivityOne");
        setResult(FragmentOne.RESULT_CODE, intent); // Вы также можете отправить
        результат без каких-либо данных, используя setResult(int resultCode)
        finish();
    }
}

```

41.18. Запуск несвязанной службы с помощью Intent

Глава 42. Фрагменты

В этой главе дается представление о фрагментах и механизме их взаимодействия.

42.1. Передача данных из активности во фрагмент с помощью объекта Bundle

Все фрагменты должны иметь пустой конструктор (то есть метод конструктора, не имеющий входных аргументов). Поэтому для передачи своих данных создаваемому фрагменту следует использовать метод setArguments(). Этот метод получает объект Bundle (бандл), в котором хранятся данные, и сохраняет его в аргументах. Впоследствии этот бандл может быть получен в вызовах onCreate() и onCreateView() фрагмента.

Активность:

```

Bundle bundle = new Bundle();
String myMessage = "Stack Overflow is cool!";
bundle.putString("message", myMessage );
FragmentClass fragInfo = new FragmentClass();
fragInfo.setArguments(bundle);

```