Курс по Android разработке

Яборов Андрей Владимирович

avyaborov@gravity-group.ru

Марквирер Владлена Дмитриевна

vdmarkvirer@hse.ru

НИУ ВШЭ 2022

GRAVITY group

Разработка под Android

Урок 3

- Типы измерений
- Ширина и высота элементов. Внутренние и внешние отступы
- Часто используемые layout
- Стили. Темы
- Переходы между окнами приложения
- Сложные элементы управления
- Base проект
- Задания

GRAVITY group

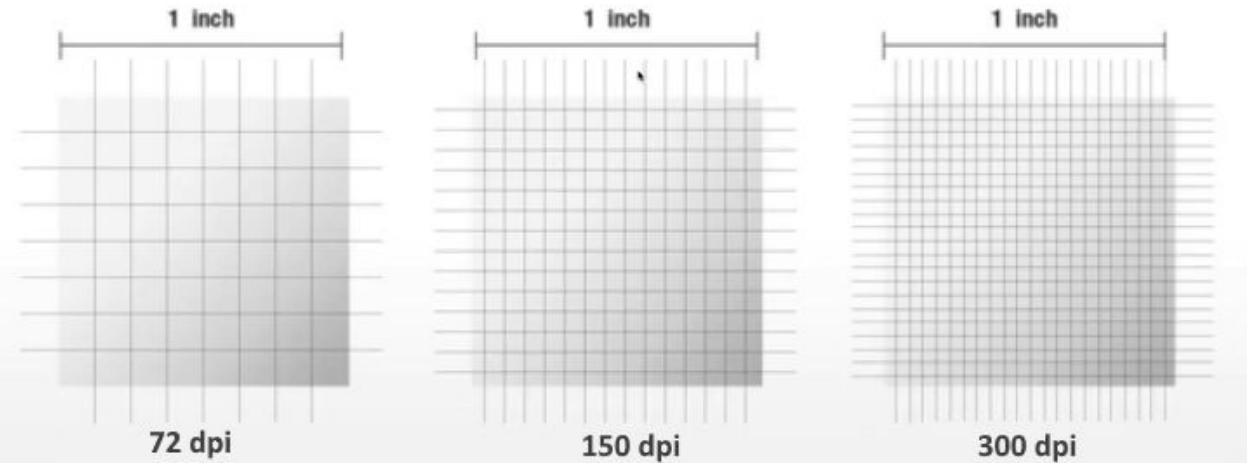
Типы измерений

В ОС Android мы можем использовать различные типы измерений:

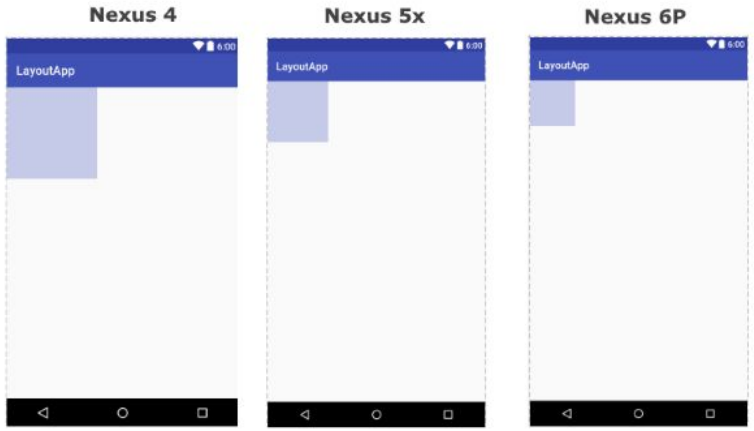
- **px**: пиксели текущего экрана. Однако эта единица измерения не рекомендуется, так как реальное представление внешнего вида может изменяться в зависимости от устройства; каждое устройство имеет определенный набор пикселей на дюйм, поэтому количество пикселей на экране может также меняться
- **dp**: (device-independent pixels) независимые от плотности экрана пиксели. Абстрактная единица измерения, основанная на физической плотности экрана с разрешением 160 dpi (точек на дюйм). В этом случае 1dp = 1px. Если размер экрана больше или меньше, чем 160dpi, количество пикселей, которые применяются для отрисовки 1dp соответственно увеличивается или уменьшается. Например, на экране с 240 dpi 1dp=1,5px, а на экране с 320dpi 1dp=2px. Общая формула для получения количества физических пикселей из dp: **px = dp * (dpi / 160)**
- **sp**: (scale-independent pixels) независимые от масштабирования пиксели. Допускают настройку размеров, производимую пользователем. Рекомендуются для работы со шрифтами.
- **pt**: 1/72 дюйма, базируются на физических размерах экрана
- **mm**: миллиметры
- **in**: дюймы

Предпочтительными единицами для использования являются **dp**. Это связано с тем, что мир мобильных устройств на Android сильно фрагментирован в плане разрешения и размеров экрана. И чем больше плотность пикселей на дюйм, тем соответственно больше пикселей нам будет доступно:

Типы измерений



Используя же стандартные физические пиксели мы можем столкнуться с проблемой, что размеры элементов также будут сильно варьироваться в зависимости от плотности пикселей устройства. Например, возьмем 3 устройства с различными характеристиками экрана Nexus 4, Nexus 5X и Nexus 6P и выведем на экран квадрат размером 300px на 300px:



В одном случае квадрат по ширине будет занимать 40%,

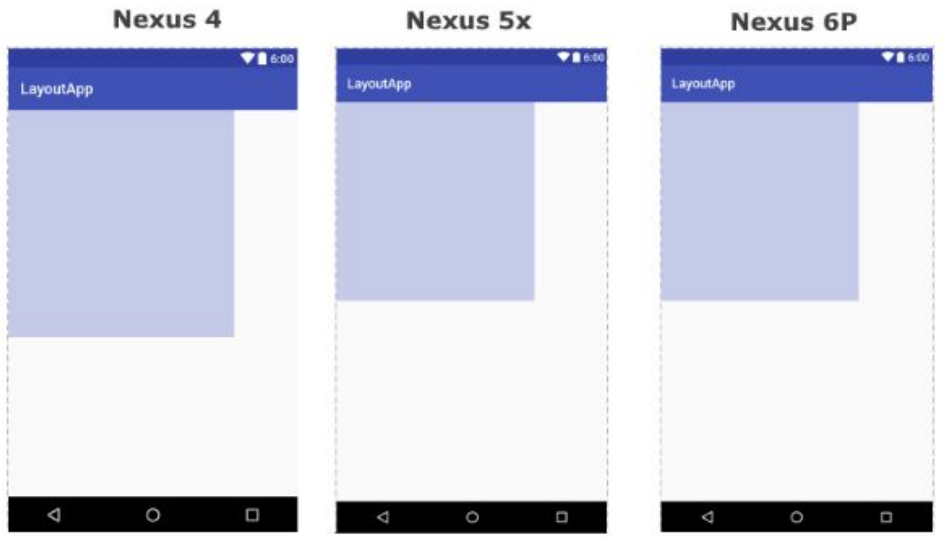
в другом - треть ширины, в третьем - 20%.

GRAVITY group

Типы измерений

Теперь также возьмем квадрат со сторонами 300x300,

но теперь вместо физических пикселей используем единицы dp:



Теперь же размеры квадрата на разных устройствах

выглядят более консистентно.

Для упрощения работы с размерами все размеры разбиты на несколько групп:

- **ldpi (low)**: ~120dpi
- **mdpi (medium)**: ~160dpi
- **hdpi (high)**: ~240dpi (к данной группе можно отнести такое устройство как Nexus One)
- **xhdpi (extra-high)**: ~320dpi (Nexus 4)
- **xxhdpi (extra-extra-high)**: ~480dpi (Nexus 5/5X, Samsung Galaxy S5)
- **xxxhdpi (extra-extra-extra-high)**: ~640dpi (Nexus 6/6P, Samsung Galaxy S6 и прочие)

GRAVITY group

Ширина и высота элементов

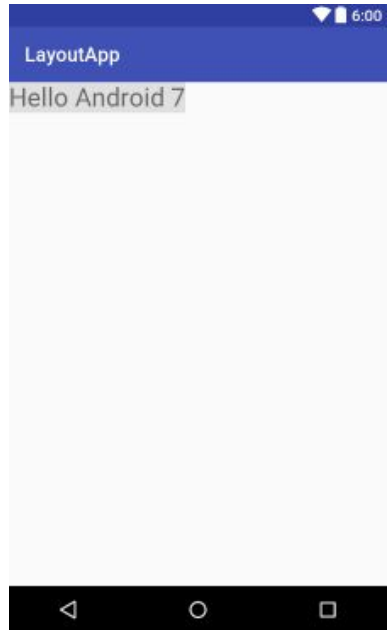
Все визуальные элементы, которые мы используем в приложении, как правило, упорядочиваются на экране с помощью контейнеров. В Android подобными контейнерами служат такие классы как RelativeLayout, LinearLayout, GridLayout, TableLayout, ConstraintLayout, FrameLayout. Все они по разному располагают элементы и управляют ими, но есть некоторые общие моменты при компоновке визуальных компонентов, которые мы сейчас рассмотрим.

Для организации элементов внутри контейнера используются параметры разметки. Для их задания в файле xml используются атрибуты, которые начинаются с префикса **layout_**. В частности, к таким параметрам относятся атрибуты **layout_height** и **layout_width**, которые используются для установки размеров и могут принимать одно из следующих значений:

- точные размеры элемента, например 96 dp
- значение **wrap_content**: элемент растягивается до тех границ, которые достаточны, чтобы вместить все его содержимое
- значение **match_parent**: элемент заполняет всю область родительского контейнера

Ширина и высота элементов

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

    <TextView
        android:text="Hello Android 7"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:textSize="26sp"
        android:background="#e0e0e0" />
</RelativeLayout>
```



Если элемент, к примеру, тот же TextView создается в коде java, то для установки высоты и ширины можно использовать метод **setLayoutParams()**:

```
textView1.setLayoutParams(new
ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT, 200));
```

В метод setLayoutParams() передается объект **ViewGroup.LayoutParams**. Этот объект инициализируется двумя параметрами: шириной и высотой. Для указания ширины и высоты можно использовать одну из констант **ViewGroup.LayoutParams.WRAP_CONTENT** или **ViewGroup.LayoutParams.MATCH_PARENT**.

GRAVITY group

Внутренние и внешние отступы

Параметры разметки позволяют задать отступы как от внешних границ элемента до границ контейнера, так и внутри самого элемента между его границами и содержимым.

Padding

Для установки внутренних отступов применяется атрибут android:padding. Он устанавливает отступы контента от всех четырех сторон контейнера. Можно устанавливать отступы только от одной стороны контейнера, применяя следующие атрибуты: android:paddingLeft, android:paddingRight, android:paddingTop и android:paddingBottom.

Margin

Для установки внешних отступов используется атрибут layout_margin. Данный атрибут имеет модификации, которые позволяют задать отступ только от одной стороны: android:layout_marginBottom, android:layout_marginTop, android:layout_marginLeft и android:layout_marginRight (отступы соответственно от нижней, верхней, левой и правой границ)

Программная установка отступов

Для программной установки внутренних отступов у элементы вызываются метод **setPadding(left, top, right, bottom)**, в который передаются четыре значения для каждой из сторон.

Для установки внешних отступов необходимо реализовать объект **LayoutParams** для того контейнера, который применяется. И затем вызвать у этого объекта LayoutParams метод **setMargins(left, top, right, bottom)**:

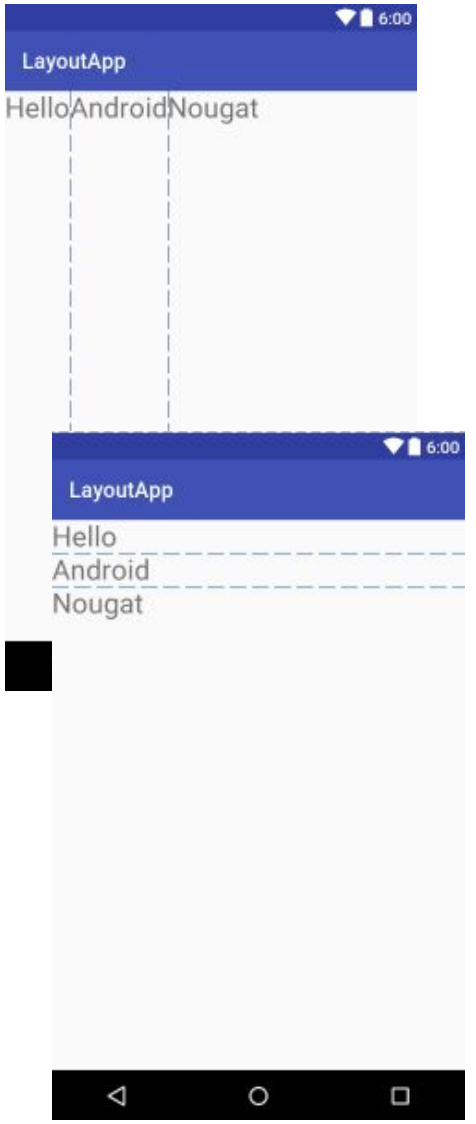
GRAVITY group

Часто используемые layout

Контейнер **LinearLayout** представляет объект ViewGroup, который упорядочивает все дочерние элементы в одном направлении: по горизонтали или по вертикали. Все элементы расположены один за другим. Направление разметки указывается с помощью атрибута **android:orientation**.

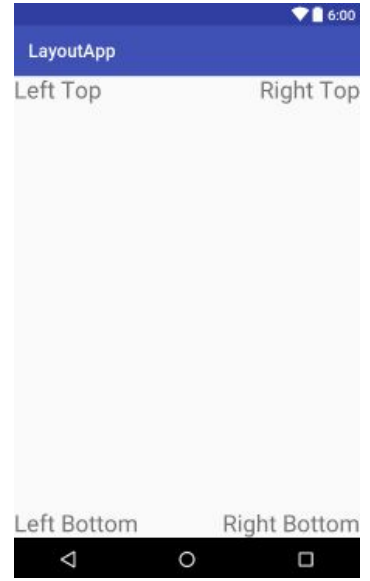
Если, например, ориентация разметки вертикальная (android:orientation="vertical"), то все элементы располагаются в столбик - по одному элементу на каждой строке. Если ориентация горизонтальная (android:orientation="horizontal"), то элементы располагаются в одну строку.

LinearLayout поддерживает такое свойство, как вес элемента, которое передается атрибутом **android:layout_weight**. Это свойство принимает значение, указывающее, какую часть оставшегося свободного места контейнера по отношению к другим объектам займет данный элемент. Например, если один элемент у нас будет иметь для свойства android:layout_weight значение 2, а другой - значение 1, то в сумме они дадут 3, поэтому первый элемент будет занимать 2/3 оставшегося пространства, а второй - 1/3. Если все элементы имеют значение android:layout_weight="1", то все эти элементы будут равномерно распределены по всей площади контейнера.



Часто используемые layout

RelativeLayout представляет объект ViewGroup, который располагает дочерние элементы относительно позиции других дочерних элементов разметки или относительно области самой разметки RelativeLayout. Используя относительное позиционирование, мы можем установить элемент по правому краю или в центре или иным способом, который предоставляет данный контейнер. Для установки элемента в файле xml мы можем применять следующие атрибуты:

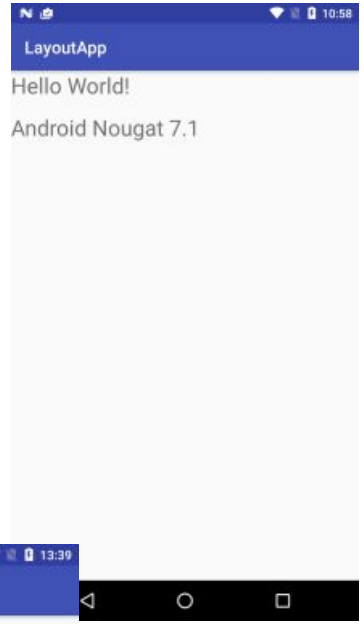


- **android:layout_above**: располагает элемент над элементом с указанным Id
- **android:layout_below**: располагает элемент под элементом с указанным Id
- **android:layout_toLeftOf**: располагается слева от элемента с указанным Id
- **android:layout_toRightOf**: располагается справа от элемента с указанным Id
- **android:layout_alignBottom**: выравнивает элемент по нижней границе другого элемента с указанным Id
- **android:layout_alignLeft**: выравнивает элемент по левой границе другого элемента с указанным Id
- **android:layout_alignRight**: выравнивает элемент по правой границе другого элемента с указанным Id
- **android:layout_alignTop**: выравнивает элемент по верхней границе другого элемента с указанным Id
- **android:layout_alignBaseline**: выравнивает базовую линию элемента по базовой линии другого элемента
- с указанным Id
- **android:layout_alignParentBottom**: если атрибут имеет значение true, то элемент прижимается к нижней
- границе контейнера
- **android:layout_alignParentRight**: если атрибут имеет значение true, то элемент прижимается к правому
- краю контейнера
- **android:layout_alignParentLeft**: если атрибут имеет значение true, то элемент прижимается к левому краю
- контейнера
- **android:layout_alignParentTop**: если атрибут имеет значение true, то элемент прижимается к верхней границе контейнера
- **android:layout_centerInParent**: если атрибут имеет значение true, то элемент располагается по центру родительского контейнера
- **android:layout_centerHorizontal**: при значении true выравнивает элемент по центру по горизонтали
- **android:layout_centerVertical**: при значении true выравнивает элемент по центру по вертикали

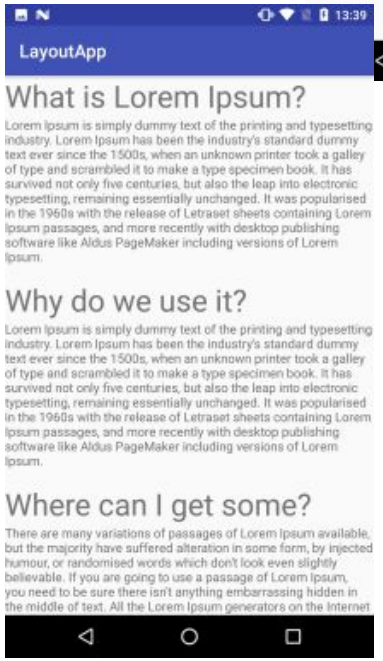
GRAVITY group

Часто используемые layout

Контейнер **FrameLayout** предназначен для вывода на экран одного помещенного в него визуального элемента. Если же мы поместим несколько элементов, то они будут накладываться друг на друга.



Контейнер **ScrollView** предназначен для создания прокрутки для такого интерфейса, все элементы которого одномоментно не могут поместиться на экране устройства. ScrollView может вмещать только один элемент, поэтому если мы хотим разместить несколько элементов, то их надо поместить в какой-нибудь контейнер.



GRAVITY group

Часто используемые layout

ConstraintLayout представляет новый тип контейнеров, который является развитием RelativeLayout и позволяет создавать гибкие и масштабируемые интерфейсы. Начиная с версии Android Studio 2.3 ConstraintLayout был добавлен в список стандартных компонентов и даже является контейнером, который используется в файлах layout по умолчанию.

Для позиционирования элемента внутри ConstraintLayout необходимо указать ограничения (constraints). Есть несколько типов ограничений. В частности, для установки позиции относительно определенного элемента используются следующие ограничения:

- **layout_constraintLeft_toLeftOf**: левая граница позиционируется относительно левой границы другого элемента
- **layout_constraintLeft_toRightOf**: левая граница позиционируется относительно правой границы другого элемента
- **layout_constraintRight_toLeftOf**: правая граница позиционируется относительно левой границы другого элемента
- **layout_constraintRight_toRightOf**: правая граница позиционируется относительно правой границы другого элемента
- **layout_constraintTop_toTopOf**: верхняя граница позиционируется относительно верхней границы другого элемента
- **layout_constraintBottom_toBottomOf**: нижняя граница позиционируется относительно нижней границы другого элемента
- **layout_constraintBaseline_toBaselineOf**: базовая линия позиционируется относительно базовой линии другого элемента
- **layout_constraintTop_toBottomOf**: верхняя граница позиционируется относительно нижней границы другого элемента
- **layout_constraintBottom_toTopOf**: нижняя граница позиционируется относительно верхней границы другого элемента
- **layout_constraintStart_toEndOf**: аналог layout_constraintLeft_toRightOf
- **layout_constraintStart_toStartOf**: аналог layout_constraintLeft_toLeftOf
- **layout_constraintEnd_toStartOf**: аналог layout_constraintRight_toLeftOf
- **layout_constraintEnd_toEndOf**: аналог layout_constraintRight_toRightOf

Позиционирования может производиться относительно границ самого контейнера ContentLayout (в этом случае ограничение имеет значение "parent"), либо же относительно любого другого элемента внутри ConstraintLayout, тогда в качестве значения ограничения указывается id этого элемента.

GRAVITY group

Стили

Стиль — это один или несколько сгруппированных атрибутов форматирования, которые отвечают за внешний вид и поведение элементов или окна. Стиль может задавать такие свойства, как ширину, отступы, цвет текста, размер шрифта, цвет фона и так далее. Сами стили хранятся в XML-файлах, отдельно от файлов разметки.

Подобное разделение напоминает использование каскадных стилей CSS для веб-документов, которые также отвечают за стили HTML-элементов и хранятся в отдельных файлах.

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:textSize="18sp"
    android:text="@string/hello" />
```

res/values/styles.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="MyTextStyle" parent="@style/Text">
        <item name="android:textSize">18sp</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

```
<TextView
    style="@style/MyTextStyle"
    ...
    android:text="@string/hello" />
```

GRAVITY group

Стили

Создать файл со стилями несложно. В проекте, создаваемом студией, уже есть готовый файл res/values/styles.xml, в который вы можете добавить новые стили. А также вы можете создать свой отдельный файл стилей.

1. Создаем новый XML-файл в папке **res/values/** вашего проекта.
2. Имя файла не имеет значения, главное, чтобы расширение было XML, а сам файл находился в указанной папке.
3. Корневым узлом файла должен быть элемент **<resources>**.
4. Для каждого элемента, которому требуется стиль, нужно добавить элемент **<style>** с уникальным именем.
5. Далее создаются элементы **<item>** для каждого свойства и присваиваются им имена, которые отвечают за выбранное свойство. Значением элемента **<item>** должно выступать ключевое слово, цвет в шестнадцатеричном значении, ссылка на другой тип ресурсов или другое значение в зависимости от свойства стиля.

Стили

Во время компиляции все свойства из файла стилей будут извлечены и применены к элементам.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="MyTextStyle" parent="@android:style/TextAppearance.Medium">
        <item name="android:layout_width">match_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

Атрибут **parent** для элемента **style** является необязательным и позволяет задавать идентификатор ресурса другого стиля, из которого нужно наследовать свойства. При этом вы можете переопределить свойства в случае необходимости.

Если наследуете новый стиль от своего стиля, то атрибут **parent** не пишется, достаточно в имени стиля указать наследуемый стиль, после которого через точку записываем новое имя стиля.

```
<style name="MyTextStyle.Red">
    <item name="android:textColor">#FF0000</item>
</style>
```

```
<style name="MyTextStyle.Red.Big">
    <item name="android:textSize">30sp</item>
</style>
```

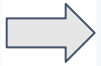
...

GRAVITY group

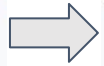
Свойства стилей (элемент item)

Для поиска свойств, которые применимы к заданному View, можно обратиться к документации и просмотреть все поддерживаемые свойства. Так все атрибуты, перечисленные в таблице атрибутов класса TextView могут быть использованы для элементов TextView или EditText. Например, у данных элементов есть свойство android:inputType:

```
<EditText
    android:inputType="number"
    ... />
```



```
<style name="Numbers">
  <item name="android:inputType">number</item>
  ...
</style>
```



```
<EditText
    style="@style/Numbers"
    ... />
```

В больших проектах повторное использование стиля поможет вам сэкономить и время и силы.

Не забывайте использовать префикс android перед именем в каждом элементе item:

<item name="android:inputType">

Для просмотра всех существующих стилей вы можете посмотреть исходники Android. Найдите папку, в которую вы устанавливали Android SDK, там можно найти нужные исходники.
Например, если есть путь к исходникам стилей Android API 17, то обращение к нему может выглядеть следующим образом:

..\platforms\android-17\data\res\values\styles.xml.

GRAVITY group

Свойства стилей (элемент item)

Помните, что все объекты View не поддерживает сразу все существующие атрибуты, поэтому используйте только специфичные стили для выбранного элемента. Но если вы по ошибке зададите ошибочный стиль для View, то это не вызовет краха приложения. Элемент View будет использовать только подходящие свойства и игнорировать чужие для него свойства.

Отдельно стоит отметить создание стиля для кнопки. У кнопки есть несколько состояний - обычное, в фокусе, нажатое, нажатое с фокусом. Поэтому для кнопки нужно создать четыре отдельных стиля, чтобы кнопка выглядела профессионально.

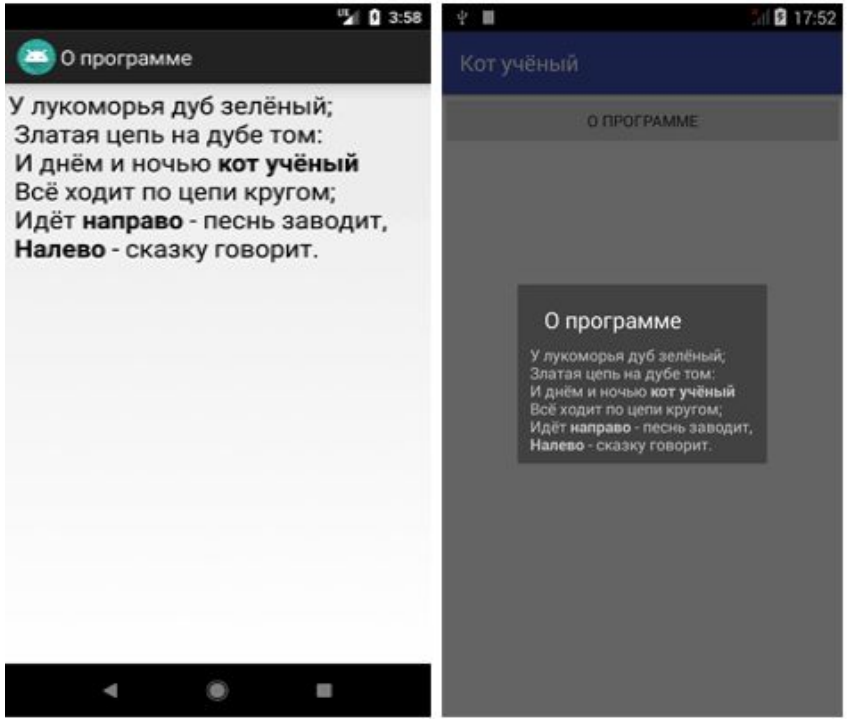
Существуют также свойства, которые не поддерживаются ни одним элементом View и применимы только как тема. Подобные стили действуют сразу на всё окно, а не на отдельный элемент. Например, есть тема, скрывающая заголовок приложения, строку состояния или изменяющая фон окна. Подобные стили легко определить по слову window, с которого начинается название стиля: **windowNoTitle, windowBackground**.

Тема

Тема - это более ёмкое понятие. По существу, тема - стиль, который относится ко всему экрану активности или приложению, а не к отдельному компоненту приложения. Таким образом, тема имеет свои атрибуты и свою область применения.
Темы похожи на определения стилей. Точно так же, как стили, темы объявляются в XML-файле элементами <style>, и ссылаются на них тем же самым способом. Различие состоит в том, что тема добавляется ко всему приложению или к отдельной активности через элементы **<application>** и **<activity>** в файле манифеста приложения, т. к. темы не могут быть применены к отдельным компонентам.

```
<application android:theme="@style/CustomTheme">
```

Если вы хотите, чтобы тема относилась не ко всему приложению, а к отдельной активности, то атрибут **android:theme** нужно добавить в тег **<activity>.**



Во многих случаях нет необходимости придумывать свои стили и темы, так как Android содержит множество собственных встроенных тем. Например, вы можете использовать тему Dialog, чтобы окно приложения выглядело как диалоговое окно.

```
<activity android:name=".About"
    android:label="@string/about_title"
    android:theme="@style/Theme.AppCompat.Dialog";>
</activity>
```

GRAVITY group

Тема

Если вам нравится тема, но несколько свойств всё-таки хотите подправить под себя, то просто добавьте тему как родительскую тему к своей теме. Например, мы хотим модифицировать стандартную тему Theme.Light, чтобы использовать свои цвета.

```
<color name="custom_theme_color">#b0b0ff</color>
<style name="CustomTheme" parent="android:Theme.Light">
    <item name="android:windowBackground">@color/custom_theme_color</item>
    <item name="android:colorBackground">@color/custom_theme_color</item>
</style>
```

Теперь мы можем использовать свой стиль вместо **Theme.Light** в манифесте:

```
<activity android:theme="@style/CustomTheme">
```

Список свойств, которые используются для настройки собственных тем:

- **android:windowNoTitle**: используйте значение true, чтобы скрыть заголовок
- **android:windowFullscreen**: используйте значение true, чтобы скрыть строку состояния и освободить место для приложения
- **android:windowBackground**: ресурс цвета или drawable для фона
- **android:windowContentOverlay**: Drawable, который рисуется поверх содержимого окна. По умолчанию, это тень от строки состояния. Можно использовать null (@null в XML-файле) для удаления ресурса.

Тема

В Android 5.0 появились новые темы, которые получили название Material Design.

- **Theme.Material** (тёмная версия)
- **Theme.Material.Light** (светлая версия)
- **Theme.Material.Light.DarkActionBar**
  (светлая версия с тёмным заголовком)

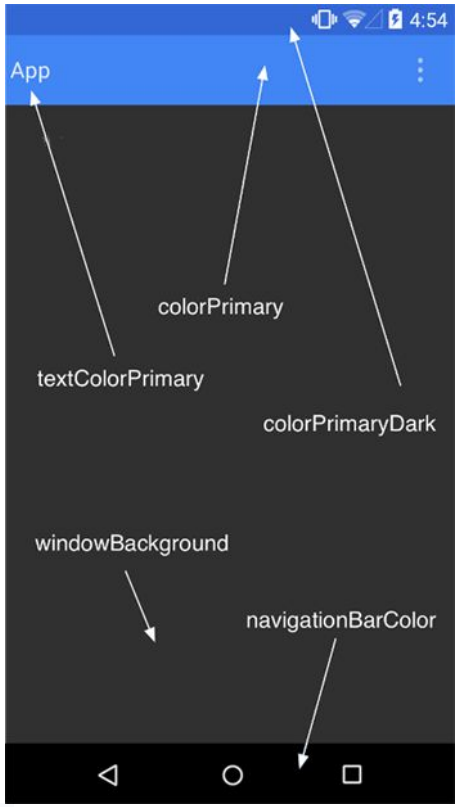
В Android 9.0 темы Material Design продолжили развитие.

- **Theme.MaterialComponents**
- **Theme.MaterialComponents.NoActionBar**
- **Theme.MaterialComponents.Light**
- **Theme.MaterialComponents.Light.NoActionBar**
- **Theme.MaterialComponents.Light.DarkActionBar**

Тема

Для Material Design были разработаны новые атрибуты тем.

- **android:colorPrimary**: основной цвет для интерфейса программы - панель, кнопки и т.д.
- **android:colorPrimaryDark**: цвет для системных элементов - строка состояния
- **android:colorAccent**: Цвет по умолчанию для компонентов, которые находятся в фокусе или активны
- **android:colorControlNormal**: Цвет для неактивных компонентов
- **android:colorControlActivated**: Цвет для активных компонентов
- **android:colorControlHighlight**: Цвет для нажатых элементов интерфейса
- **colorSwitchThumbNormal**: и т.д. изучаем документацию



Позже были добавлены другие атрибуты:

colorPrimaryVariant, colorOnPrimary, colorSecondary, colorSecondaryVariant, colorOnSecondary, colorError, colorOnError, colorSurface, colorOnSurface, colorBackground, colorOnBackground.

```
<color name="primaryColor">#FF9800</color>
<color name="primaryColorDark">#F57C00</color>
```

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/primaryColor</item>
        <item name="colorPrimaryDark">@color/primaryColorDark</item>
    </style>

</resources>
```

GRAVITY group

Тема

Третий важный цвет для использования в приложениях - акцентированный. Данный цвет может использоваться для кнопки Floating Action Button и для различных компонентов. Он должен быть достаточно контрастным по сравнению с основным цветом.

Акцентированные цвета поддерживаются многими компонентами из коробки. Для некоторых следует использовать аналоги из библиотеки AppCompat:

- Флажки и переключатели
- SwitchCompat вместо Switch
- Курсор у EditText
- Текст у TextInputLayout
- Текущий индикатор у TabLayout
- Выбранный элемент у NavigationView
- Фон у FloatingActionButton

```
<color name="accentColor">#00E676</color>
```

```
<item name="colorAccent">@color/accentColor</item>

<item name="android:colorAccent">@color/accentColor</item>
```

Переходы между окнами приложения

Простое переключение на другой экран

Приложение не всегда состоит из одного экрана. Например, мы создали очень полезную программу и пользователю хочется узнать, кто же её автор. Он нажимает на кнопку «О программе» и попадает на новый экран, где находится полезная информация о версии программы, авторе, адресе сайта и т.д. Воспринимайте экран активности как веб-страницу с ссылкой на другую страницу.

В проекте 0 мы уже создавали вторую активность и соответствующую разметку экрана, а также переключали экраны в манифесте для запуска нужной нам активности. Теперь научимся открывать нужный нам экран по нажатию на кнопку.

Переходим к классу MainActivity (или любому другому классу, который описывает поведение запускаемого в самом начале экрана). Напишем обработчик щелчка кнопки:

```
public void onClick(View view) {
    Intent intent = new Intent(MainActivity.this, AboutActivity.class);
    startActivity(intent);
}
```

Для запуска нового экрана необходимо создать экземпляр класса **Intent** и указать в первом параметре текущий класс, а во втором - класс для перехода, например, AboutActivity. После этого вызывается метод **startActivity()**, который и запускает новый экран.

Переходы между окнами приложения

Простое переключение на другой экран

После вызова метода **startActivity()** запустится новая активность (в данном случае AboutActivity), она станет видимой и переместится на вершину стека, содержащего работающие компоненты. При вызове метода **finish()** из новой активности (или при нажатии аппаратной клавиши возврата) она будет закрыта и удалена из стека. Разработчик также может перемещаться к предыдущей (или к любой другой) активности, используя всё тот же метод **startActivity()**.

Нужно помнить, что необходимо **зарегистрировать** новый Activity в манифесте AndroidManifest.xml. Найдите этот файл в своем проекте и дважды щёлкните на нём. Откроется окно редактирования файла. Добавьте новый тег **<activity>** после закрывающего тега **</activity>** для первой активности. Печатайте самостоятельно и активно используйте подсказки. Получится примерно следующее:

```
<activity android:name=".AboutActivity"
    android:label="@string/about_title">
</activity>
```

Не забывайте, что второй создаваемый класс активности должен:

1. Наследоваться от класса Activity или ему похожих (ListActivity и др.).
2. Иметь XML-файл разметки (если требуется).
3. Быть прописан в манифесте.

Переходы между окнами приложения

Передача данных между активностями

Мы использовали простейший пример для вызова другого экрана активности. Иногда требуется не только вызвать новый экран, но и передать в него данные. Например, имя пользователя. В этом случае нужно задействовать специальную область **extraData**, который имеется у класса **Intent**.

Область extraData - это список пар ключ/значение, который передаётся вместе с Intent. В качестве ключей используются строки, а для значений можно использовать любые примитивные типы данных, массивы примитивов, объекты класса Bundle и др.

Для передачи данных в другую активность используется метод **putExtra()**: `intent.putExtra("Ключ", "Значение");`

intent.putExtra("username", userEditText.getText().toString());

Принимающая активность должна вызвать какой-нибудь подходящий метод: **getIntExtra()**, **getStringExtra()** и т.д.:

```
int count = getIntent().getIntExtra("name", 0);
```

В обработчике нажатия кнопки между созданием **Intent** и **startActivity()**, необходимо указать что нужно передать во вторую активность через **putExtra()** - может быть несколько значений, а во второй активности в методе **onCreate()** получаем данные, которые можно далее использовать, например:

String user = getIntent().getExtras().getString("username");

GRAVITY group

Сложные элементы управления

Spinner - выпадающий список

Компонент **Spinner** из раздела Containers (раньше был в разделе Widgets) похож на выпадающий список, используемый в ОС Windows. В закрытом состоянии компонент показывает одну строчку, при раскрытии выводит список в виде диалогового окна с переключателями.

При добавлении элемента на экран отображается просто полоска со строкой **Item1**.
В основном настройка происходит программным путем. Но можно и через XML.
Добавим в строковый файл ресурсов **strings.xml** несколько элементов массива:

```
<string-array name="catNames">
    <item>Барсик</item>
    <item>Мурзик</item>
    <item>Васька</item>
    <item>Рыжик</item>
</string-array>
```

Теперь осталось в атрибуте **android:entries** указать на созданный массив и компонент Spinner будет заполнен данными.

```
<Spinner
    android:id="@+id/spinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/catNames" />
```

Если нужно из программы узнать, какой пункт из выпадающего списка выбран в Spinner, то можно использовать такой код, например, при нажатии кнопки:

```
Spinner spinner = findViewById(R.id.spinner);
String selected = spinner.getSelectedItem().toString();
Toast.makeText(getApplicationContext(), selected, Toast.LENGTH_SHORT).show();
```

Если нужен не текст, а номер позиции, то вызывайте метод **getSelectedItemPosition()**

GRAVITY group

Сложные элементы управления

Spinner - выпадающий список

Spinner (как, например, и компонент ListView) использует адаптер данных для связывания содержимого из набора данных с каждым пунктом в списке. Для загрузки данных нужно:

1. Получить экземпляр компонента Spinner
2. Настроить адаптер данных для связывания
3. Вызвать метод **setAdapter()**

Данные в закрытом и раскрытом состоянии Spinner отображает по разному. Поэтому необходимо создавать макеты шаблонов для обоих состояний. Android предоставляет несколько своих собственных ресурсов для Spinner для простых задач. Например, есть ресурс **android.R.layout.simple_spinner_item** для создания представления для каждого элемента списка. Ресурс **android.R.layout.simple_spinner_dropdown_item** служит шаблоном для раскрывающегося списка.

Создадим строковый массив в файле **strings.xml**:

```
<string-array name="animals">
    <item>Кот</item>
    <item>Кошка</item>
    <item>Котёнок</item>
    <item>Животное</item>
</string-array>
```

Сложные элементы управления

Spinner - выпадающий список

Загрузим строковый массив с именем animals в экземпляр класса ArrayAdapter при помощи метода **createFromResource()**:

```
// Получаем экземпляр элемента Spinner
final Spinner spinner = findViewById(R.id.spinner);

// Настраиваем адаптер
ArrayAdapter<?> adapter =
        ArrayAdapter.createFromResource(this, R.array.animals, android.R.layout.simple_spinner_item);
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

// Вызываем адаптер
spinner.setAdapter(adapter);
```

По умолчанию выводится первый элемент списка. С помощью метода **setSelection()** можно установить нужный элемент по умолчанию, указав индекс из строкового ресурса.

```
spinner.setSelection(2);
```

Сложные элементы управления

Spinner - выпадающий список

Если вам нужно получить выбранный элемент сразу в момент выбора, то используйте метод **setOnItemSelectedListener()** и реализовать метод **onItemSelected()** класса **AdapterView.OnItemSelectedListener.**

```
spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
        public void onItemSelected(AdapterView<?> parent,
                        View itemSelected, int selectedItemPosition, long selectedId) {

                String[] choose = getResources().getStringArray(R.array.animals);
                Toast toast = Toast.makeText(getApplicationContext(),
                                "Ваш выбор: " + choose[selectedItemPosition], Toast.LENGTH_SHORT);
                toast.show();
        }
        public void onNothingSelected(AdapterView<?> parent) {
        }
});
```

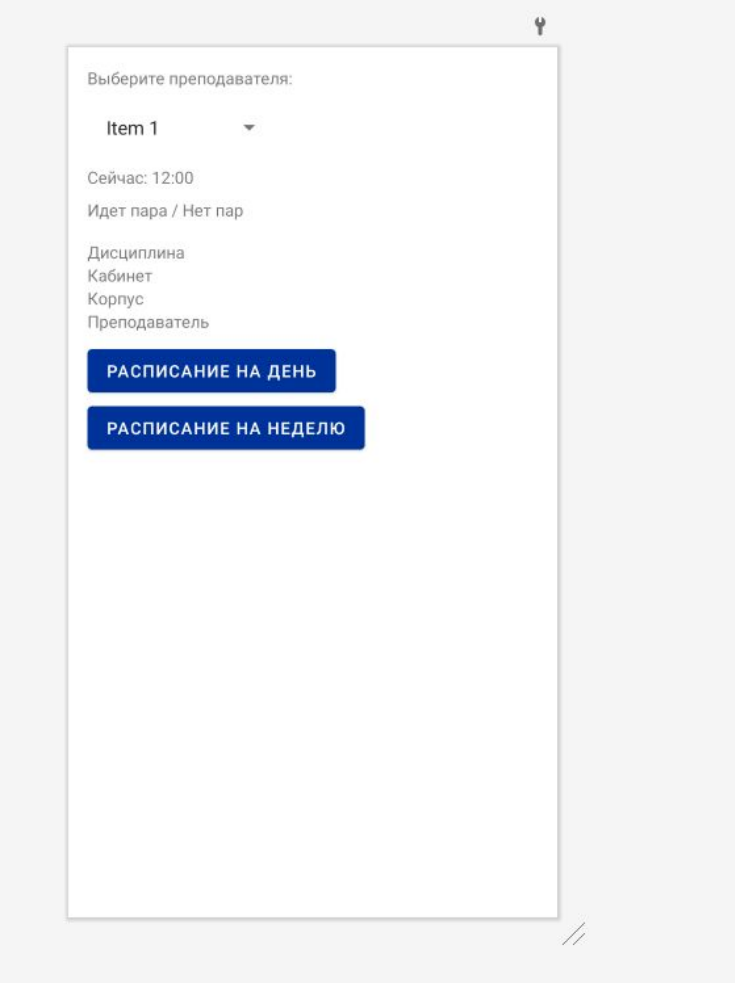
Теперь при выборе любого пункта вы получите всплывающее сообщение о выбранном пункте. Обратите внимание, что нам также пришлось реализовать вызов обратного вызова **onNothingSelected()**.

Base проект


```java
public class MainActivity extends AppCompatActivity {

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

        View buttonStudent = findViewById(R.id.button_student);
        View buttonTeacher = findViewById(R.id.button_teacher);

        buttonStudent.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                showStudent();
            }
        });

        buttonTeacher.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                showTeacher();
            }
        });
    }

    private void showStudent() {
        Intent intent = new Intent( packageContext: this, StudentActivity.class);
        startActivity(intent);
    }

    private void showTeacher() {
        Intent intent = new Intent( packageContext: this, TeacherActivity.class);
        startActivity(intent);
    }
}
```

GRAVITY group

Base проект



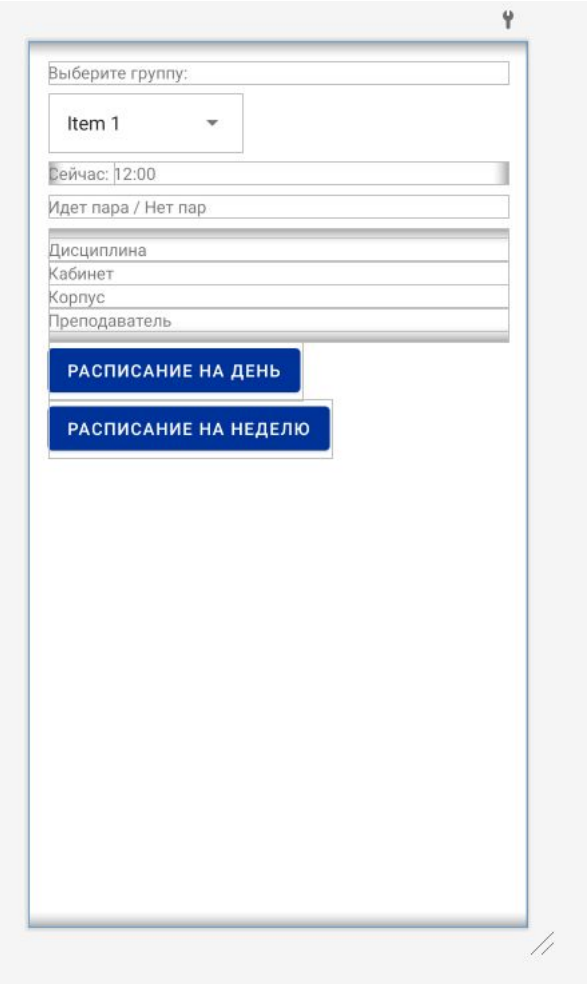

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_teacher);

    final Spinner spinner = findViewById(R.id.groupList);

    List<StudentActivity.Group> groups = new ArrayList<>();
    initGroupList(groups);

    ArrayAdapter<?> adapter = new ArrayAdapter<>( context: this, android.R.layout.simple_spinner_item, groups);
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

    spinner.setAdapter(adapter);

    spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
        public void onItemSelected(AdapterView<?> parent,
                                   View itemSelected, int selectedItemPosition, long selectedId) {
            Object item = adapter.getItem(selectedItemPosition);
            Log.d(TAG, msg: "selectedItem: " + item);
        }

        public void onNothingSelected(AdapterView<?> parent) {
            //
        }
    });

    time = findViewById(R.id.time);
    initTime();

    status = findViewById(R.id.status);
    subject = findViewById(R.id.subject);
    cabinet = findViewById(R.id.cabinet);
    corp = findViewById(R.id.corp);
    teacher = findViewById(R.id.teacher);

    initData();
}
```

GRAVITY group

Base проект

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

    final Spinner spinner = findViewById(R.id.groupList);

    List<Group> groups = new ArrayList<>();
    initGroupList(groups);

    ArrayAdapter<?> adapter = new ArrayAdapter<>( context: this, android.R.layout.simple_spinner_item, groups
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

    spinner.setAdapter(adapter);

    spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
        public void onItemSelected(AdapterView<?> parent,
                                   View itemSelected, int selectedItemPosition, long selectedId) {
            Object item = adapter.getItem(selectedItemPosition);
            Log.d(TAG,  msg: "selectedItem: " + item);
        }

        public void onNothingSelected(AdapterView<?> parent) {
            //
        }
    });

    time = findViewById(R.id.time);
    initTime();

    status = findViewById(R.id.status);
    subject = findViewById(R.id.subject);
    cabinet = findViewById(R.id.cabinet);
    corp = findViewById(R.id.corp);
    teacher = findViewById(R.id.teacher);

    initData();
}
```

```java
////////////////////////////////
static class Group {

    private Integer id;
    private String name;

    public Group(Integer id, String name) {
        this.id = id;
        this.name = name;
    }

    public Integer getId() {
        return id;
    }

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

    @Override
    public String toString() {
        return name;
    }

    public String getName() {
        return name;
    }

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



Base проект

Для студента

```
private void initGroupList(List<Group> groups) {
    groups.add(new Group( id: 1,  name: "ПИ-18-1"));
    groups.add(new Group( id: 2,  name: "ПИ-18-2"));
}

private void initTime() {
    currentTime = new Date();
    SimpleDateFormat simpleDateFormat = new SimpleDateFormat( pattern: "HH:mm", Locale.getDefault());
    time.setText(simpleDateFormat.format(currentTime));
}

private void initData() {
    status.setText("Нет пар");

    subject.setText("Дисциплина");
    cabinet.setText("Кабинет");
    corp.setText("Корпус");
    teacher.setText("Преподаватель");
}
```

GRAVITY group

Base проект

Для преподавателя

```
private void initGroupList(List<StudentActivity.Group> groups) {
    groups.add(new StudentActivity.Group( id: 1,  name: "Преподаватель 1"));
    groups.add(new StudentActivity.Group( id: 2,  name: "Преподаватель 2"));
}

private void initTime() {
    currentTime = new Date();
    SimpleDateFormat simpleDateFormat = new SimpleDateFormat( pattern: "HH:mm", Locale.getDefault());
    time.setText(simpleDateFormat.format(currentTime));
}

private void initData() {
    status.setText("Нет пар");

    subject.setText("Дисциплина");
    cabinet.setText("Кабинет");
    corp.setText("Корпус");
    teacher.setText("Преподаватель");
}
```

GRAVITY group

Base проект

Задания

1. Доработать функцию формирования списка групп `StudentActivity#initGroupList` Сделать ее динамической на основе шаблона (образовательная программа-год поступления-номер группы), данные для шаблона взять из данных ВШЭ.
2. Добавить стили для экранов. Вынести в файл со стилями стили для всех лейблов и значений, сделать лейблы жирным стилем и цвета **`colorSecondary`**.
3. Доработать функцию `initTime`. Вывести время и текущий день в текстовом виде в русской локале (Пример: 12:00, Понедельник)
4. Расположить кнопки с расписанием в один ряд и сделать надписи в 2 строки на них

Литература

https://developer.android.com/guide/topics/ui/declaring-layout

https://developer.android.com/guide/topics/ui/look-and-feel

https://material.io/develop/android/theming/color

https://developer.android.com/guide/topics/ui/controls