**Стили и темы**

Откройте снова файл манифеста **AndroidManifest.xml** из прошлого урока и измените строчку для активности **AboutActivity**, указав тему.

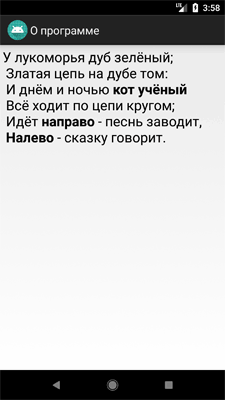
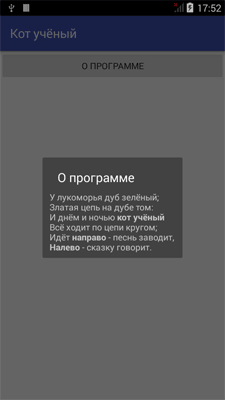
<activity android:name=".About"

android:label="@string/about\_title"

android:theme="@style/Theme.Material3.DayNight.Dialog">

</activity>

Запустив программу, вы увидите, что внешний вид окна «О программе» стал уже другим. Сравните.

Обратите внимание, что теперь появляется не окно во весь экран, а диалоговое окно в центре экрана. При этом остальная часть экрана затемняется.

На картинках представлен внешний вид экранов активностей от древних телефонов, сейчас темы выглядят по другому на ваших телефонах.

Тему можно применить не только к отдельной активности, но и ко всем активностям приложения, если прописать в теге **application**.

Кстати, вы можете разработать свою тему на основе существующих и сохранить её в файле **res/values/styles.xml**.

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

Предположим, у нас есть следующий код разметки для **TextView**:

<TextView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:textColor="#00FF00"

android:typeface="monospace"

android:textSize="18sp"

android:text="@string/hello" />

Мы можем вынести все свойства в файл стилей следующим образом:

res/values/styles.xml

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

<resources>

<style name="MyTextStyle" parent="@style/Base.Theme.Litert>

<item name="android:textSize">18sp</item>

<item name="android:textColor">#00FF00</item>

<item name="android:typeface">monospace</item>

</style>

</resources>

Тогда в файле разметки теперь будет так:

<TextView

style="@style/MyTextStyle"

...

android:text="@string/hello" />

Как видите, мы удалили все свойства для текста из файла разметки и разместили их в файле стилей в ресурсе под именем **MyTextStyle**, который содержит теперь все необходимые свойства.

Создать файл со стилями несложно. Создаём новый XML-файл в папке **res/values/** вашего проекта. Имя файла не имеет значения, главное, чтобы расширение было XML, а сам файл находился в указанной папке. 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.

Корневым узлом файла должен быть элемент **<resources>**. Для каждого элемента, которому требуется стиль, нужно добавить элемент **<style>** с уникальным именем. 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. Ниже представлен образец такого стиля:

<?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>

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

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.

**Быстрое создание стилей через Android Studio**

Если вы решили вынести некоторые свойства компонентов в стили, то необязательно это делать вручную. В студии есть удобный инструмент для быстрого создания ресурсов стилей.

В текстовом режиме щёлкните правой кнопкой мыши на нужном компоненте и в контекстном меню выбирайте пункт **Refactor | Extract | Style**. Далее укажите имя стиля и выберите требуемые параметры для экспорта. Студия самостоятельно создаст стиль в файле **styles.xml** и автоматически применит созданный ресурс в layout-файле.

**Наследование стилей**

Наследование - мощный и полезный механизм, позволяющий не изобретать велосипед, а использовать готовые проверенные наработки. С помощью атрибута **parent** в элементе **style** вы можете наследовать нужные свойства из существующих стилей, а также переопределить некоторые свойства или добавить свои дополнительные свойства. Предположим, мы решили наследоваться от существующего системного стиля Android для текстовых сообщений и слегка модифицировать его.

<style name="GreenText" parent="@android:style/TextAppearance">

<item name="android:textColor">#00FF00</item>

</style>

Если вы собираетесь наследоваться от собственных стилей, то использовать атрибут **parent** не нужно. Просто используйте префикс имени наследуемого стиля перед создаваемым новым стилем, разделяя имена стилей точкой. Например, для создания нового стиля, который наследуется от стиля **MyTextStyle**, созданного нами ранее, где мы хотим получить красный текст, используйте следующий способ:

<style name="MyTextStyle.Red">

<item name="android:textColor">#FF0000</item>

</style>

Как видите, нам не пришлось использовать атрибут **parent** в теге **style**, потому что имя стиля начинается с имени **MyTextStyle** (созданный нами стиль). Теперь наш стиль наследует все свойства от стиля родителя, при этом мы изменили одно свойство **android:textColor**, чтобы текст выводился красным цветом. Вы можете ссылаться на новый стиль через конструкцию **@style/MyTextStyle.Red**.

Вы можете продолжать наследоваться дальше много раз, добавляя новые имена через точку. Допустим, мы хотим расширить свойство **MyTextStyle.Red**, чтобы текст выводился более крупным шрифтом:

<style name="MyTextStyle.Red.Big">

<item name="android:textSize">30sp</item>

</style>

Итак, мы наследовались от стилей **MyTextStyle** и **MyTextStyle.Red**, а затем добавили новое свойство **android:textSize**.

Не забывайте, что данная техника наследования применима только к собственным стилям. Для наследования системных стилей типа **TextAppearance** необходимо использовать атрибут **parent**.

**Свойства стиля**

Разобравшись с созданием стилей, рассмотрим различные свойства, определяемые в элементе **item**. Мы уже встречались с такими свойствами, как **layout\_width** и **textColor**. На самом деле свойств гораздо больше.

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"

... />

Но вместо этого мы можем также создать стиль для элемента **EditText**, который будет включать в себя данное свойство:

<style name="Numbers">

<item name="android:inputType">number</item>

...

</style>

В файле разметки теперь можно написать так:

<EditText

style="@style/Numbers"

... />

На первый взгляд вам возможно покажется, что подобный подход отнимает слишком много времени. Но в больших проектах повторное использование стиля поможет вам сэкономить и время и силы.

## 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.

Чтобы установить тему, откройте файл **AndroidManifest.xml** и отредактируйте тег **<application>**, чтобы он включал в себя атрибут **android:theme** с указанием имени стиля:

<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 (Смотри выше).

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">

Небольшой 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.

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

* **@android:style/Theme.Material** (тёмная версия)
* **@android:style/Theme.Material.Light** (светлая версия)
* **@android:style/Theme.Material.Light.DarkActionBar** (светлая версия с тёмным заголовком)

В Android 9.0 темы Material Design продолжили развитие, они будут активно внедряться в ближайшее время.

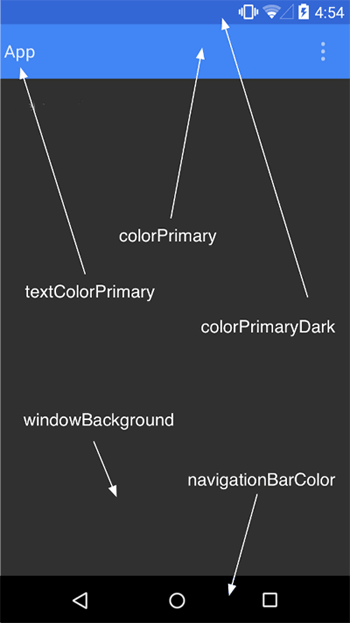
* Theme.MaterialComponents
* Theme.MaterialComponents.NoActionBar
* Theme.MaterialComponents.Light
* Theme.MaterialComponents.Light.NoActionBar
* Theme.MaterialComponents.Light.DarkActionBar

Сейчас в студии по умолчанию используются новые темы Material 3.

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**.



Настройка цветов происходит по определённым правилам. На сайте http://www.google.com/design/spec/style/color.html# (сейчас сайт переехал на другой адрес) есть таблица цветов. Обратите внимание на числа слева. Основным цветом (colorPrimary) считается цвет под номером **500**, он идёт первым в таблицах. Этот цвет должен использоваться в качестве заголовка (**Toolbar**).

Допустим, мы делаем специальное приложение для рыжего кота. Создадим новый файл **res/values/colors.xml**. На указанном сайте находим таблицу цветов оранжевого цвета **Orange** и будем использовать предлагаемое значение.

Зададим основной цвет.

<color name="primaryColor">#FF9800</color>

Для строки состояние, которая находится выше заголовка приложения, нужно использовать цвет со значением **700** (colorPrimaryDark). Это более тёмный цвет и позволяет различать заголовок приложения и строку состояния. Возвращаемся к оранжевой таблице цветов, запоминаем значение цвета и прописываем его в ресурсах.

<color name="primaryColorDark">#F57C00</color>

Пропишем в теме приложения новые элементы.

<resources>

<!-- Base application theme. -->

<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">

<!-- Customize your theme here. -->

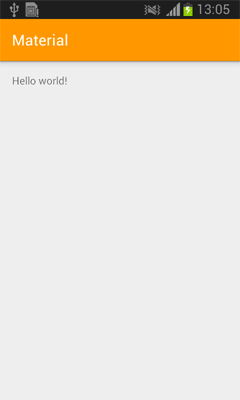
<item name="colorPrimary">@color/primaryColor</item>

<item name="colorPrimaryDark">@color/primaryColorDark</item>

</style>

</resources>

На старых устройствах цвет строки состояния не изменяется. Цвет заголовка поменять можно.



В файле **res/values-v21/styles.xml** для новых устройств нужно повторить указанные действия с небольшой поправкой уже есть предопределённые константы для эти цветов, поэтому используем в именах **android:colorPrimary** и **android:colorPrimaryDark**.

В одной из недавних версий библиотеки совместимости была реализовав **colorControlActivated** на поддержка нового дизайна для всех устройств и такое разделение сейчас не является обязательным.

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

<resources>

<style name="AppTheme" parent="android:Theme.Material.Light">

<item name="android:colorPrimary">@color/primaryColor</item>

<item name="android:colorPrimaryDark">@color/primaryColorDark</item>

</style>

</resources>

С главным цветом есть одна тонкость. Старые устройства используют **ActionBar** и его цвет подчиняется правилам Material Design из коробки. На новых устройствах для активности используется тема без панели действий **Theme.AppCompat.NoActionBar** и вручную добавляется компонент **Toolbar**. Чтобы он использовал основной цвет, используйте правильный стиль для фонового цвета.

<android.support.v7.widget.Toolbar

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:background="?attr/colorPrimary" />

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. Для примера выберем зелёный цвет по цвету глаз рыжих котов. Находим в таблице зелёный цвет и выбираем нужное значение из **A400**

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

Прописываем цвет в обоих темах:

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

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

Сейчас акцентированный цвет мы нигде не увидим. Вернёмся к нему позже.

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

Пользуйтесь сервисом [Material Design Color Palette Generator](http://www.materialpalette.com/deep-orange/light-green) для создания палитры в стиле Material: выбираем основной цвет, цвет «плавающей» кнопки и сайт генерирует необходимую палитру.

В Android 5.0 появился новый атрибут темы **colorEdgeEffect**. Вам необходимо переопределить тему, а затем применить к компоненту.

**res/values/themes.xml**

<style name="RedThemeOverlay" parent="android:ThemeOverlay.Material">

<item name="android:colorEdgeEffect">#FF0000</item>

</style>

res/layout/fragment\_list.xml

<ListView

...

android:theme="RedThemeOverlay" />

**Задание 1. Напишите и примените свою тему**

**Задание 2. Работа с ToolBar**

По умолчанию базовая ToolBar отключена, это делается в файле themes

необходимо там поменять parent на

<style name="Base.Theme.Menu" parent="Theme.Material3.DayNight">

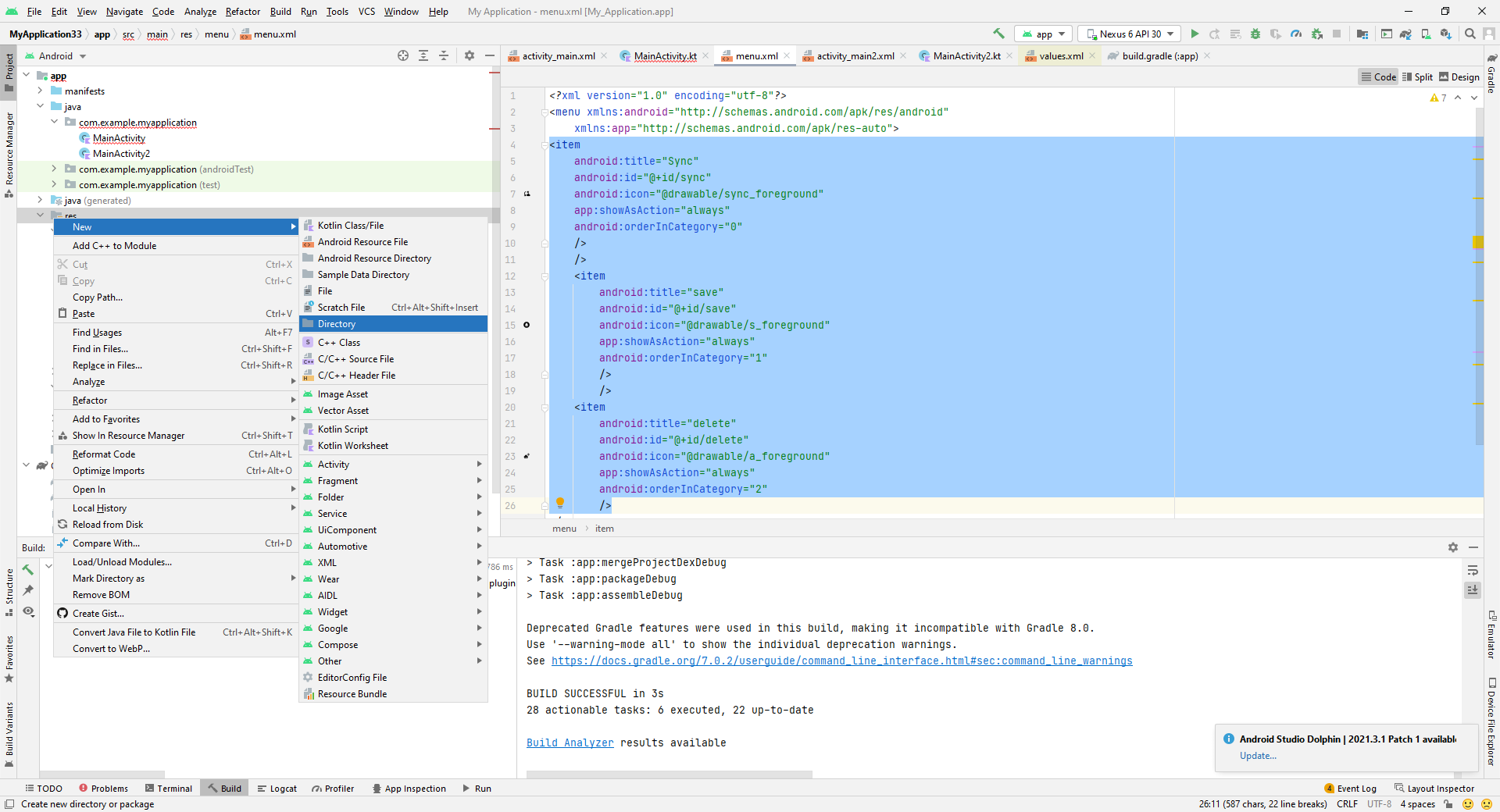
Определение кнопки возврата на ToolBar

override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
 *supportActionBar*?.setDisplayHomeAsUpEnabled(true)  
}

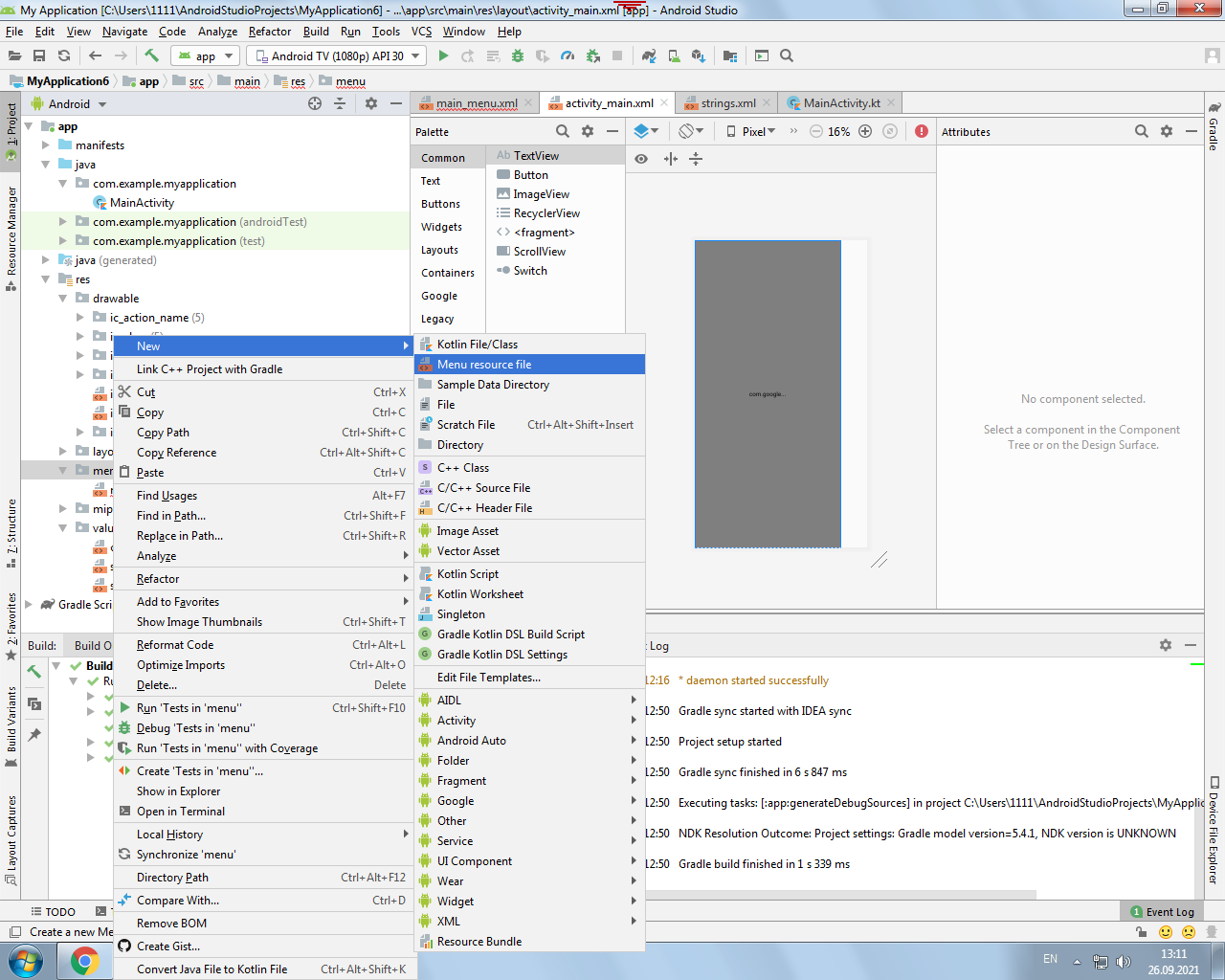
Определение действия при нажатии на кнопку

override fun onOptionsItemSelected(item: MenuItem): Boolean {  
  
 finish()  
  
 return true  
}

Создание папки для меню

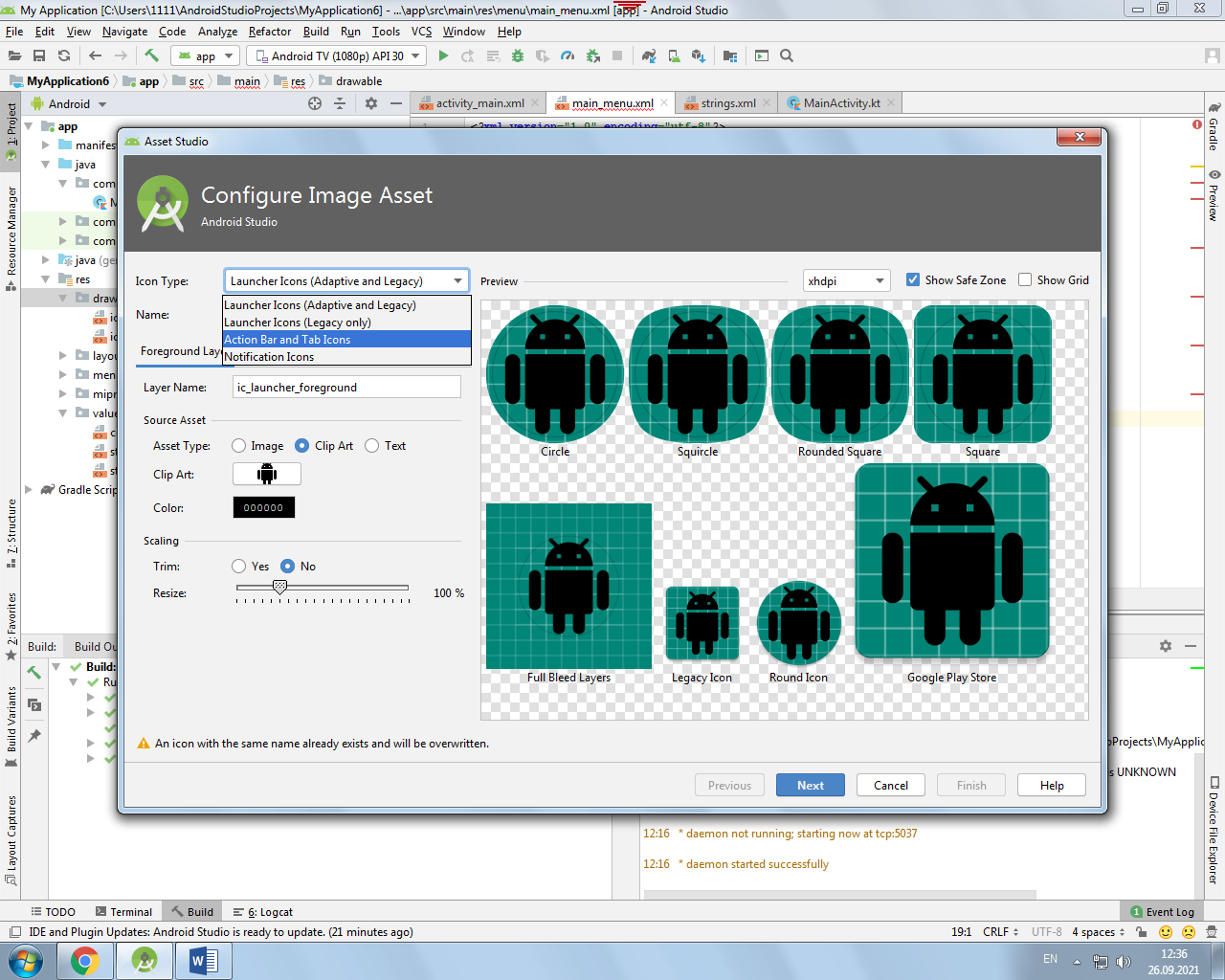


Создание файла меню в этой папке



Формирование ресурсов для меню

- 3 Иконок (имена можно давать произвольно, но не забудьте потом их при описании на свои поменять)



Формирования списка меню в файле меню

<item  
 android:title="Sync"  
 android:id="@+id/sync"  
 android:icon="@drawable/sync\_foreground"  
 app:showAsAction="always"  
 android:orderInCategory="0"  
 />  
 />  
 <item  
 android:title="save"  
 android:id="@+id/save"  
 android:icon="@drawable/s\_foreground"  
 app:showAsAction="always"  
 android:orderInCategory="1"  
 />  
 />  
 <item  
 android:title="delete"  
 android:id="@+id/delete"  
 android:icon="@drawable/a\_foreground"  
 app:showAsAction="always"  
 android:orderInCategory="2"  
 />

Перенести на форму компонент ToolBar , привязать его к форме и дать ему id=tb

Подключение меню к ToolBar через свойство menu

проверяем подключен ли файле сборки gradle (Module)

buildFeatures**{** viewBinding=true  
**}**

и прописываем в файле MainActivity в событии оnCreate

перед событием оnCreate

lateinit var binding: ActivityMainBinding

в событии оnCreate

binding = ActivityMainBinding.inflate(*layoutInflater*)  
setContentView(binding.*root*)  
  
binding.tb.setOnMenuItemClickListener()**{** when (**it**.*itemId*) {  
 android.R.id.*home* -> finish()  
 R.id.*sync* -> {  
 Toast.makeText(this, "Sync", Toast.*LENGTH\_SHORT*).show()  
 }  
 R.id.*save* -> {  
  
 Toast.makeText(this, "Save", Toast.*LENGTH\_SHORT*).show()  
 }  
  
 R.id.*delete* -> {  
 Toast.makeText(this, "Delete", Toast.*LENGTH\_SHORT*).show()  
 }  
 }  
 true  
  
**}**