Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **1**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Структура Android-приложения. Знакомство с понятиями «активность», «разметка», «ресурсы», «эмулятор». Создания первого Android-приложения.**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

# Проект 1. Разработка первого Android-приложения

**Цель работы:**

* Знакомство с Android Studio – средой быстрой разработки Android-приложений
* Знакомство с понятиями: ресурсы, разметка, активность, эмулятор, виртуальное устройство
* Привязка разметки к активности
* Написание класса, соответствующего активности
* Создание программного объекта, соответствующего элементу разметки
* Написание простейшего обработчика событий
* Привязка обработчика событий к элементу управления в коде

**Задание:**

Внешний вид активности

при запуске после нажатия на кнопки

**\*\*\***

**Btn3**

**\*\*\***

**Btn1**

**Btn1**

**Btn2**

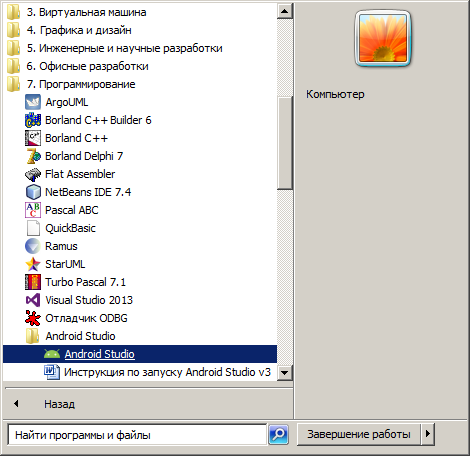
**Btn3**

**Btn4**

По нажатию на любую кнопку текст на этой кнопке должен поменяться на «\*\*\*». Использовать один обработчик события. Привязка обработчика событий к элементу управления должна осуществляться в коде, а не в разметке. Для надписей на кнопках использовать ссылки на строковые ресурсы.

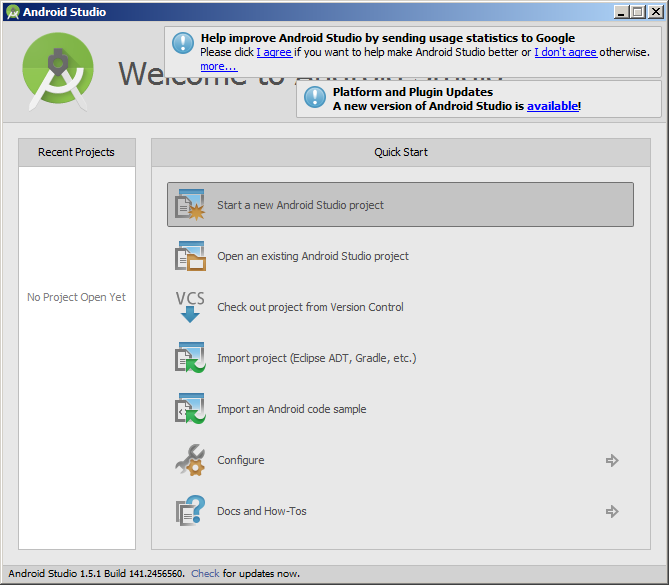
**Указания к выполнению задания:**

Запуск Android Studio производим с установками, для которых имеется достаточное количество ресурсов на компьютерах вычислительного центра Радиополитехникума (2016г.).

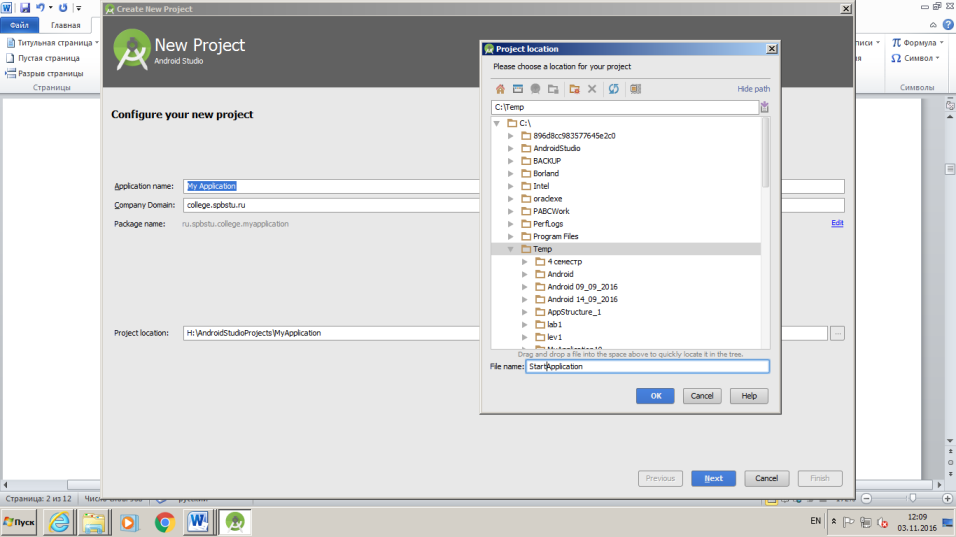


Начинаем запуск щелчком по соответствующему пункту меню.

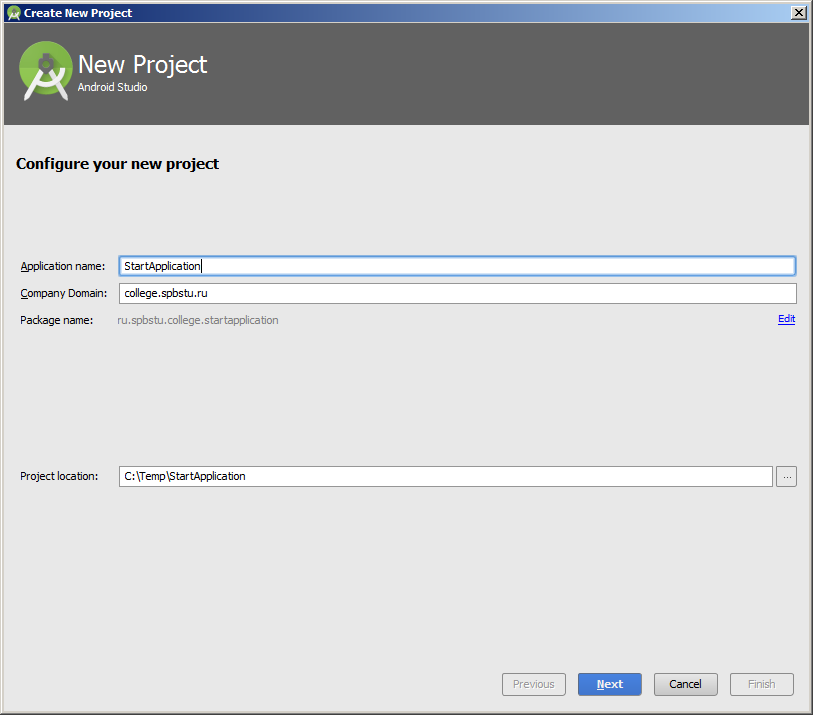
После этого мы увидим следующее окно:



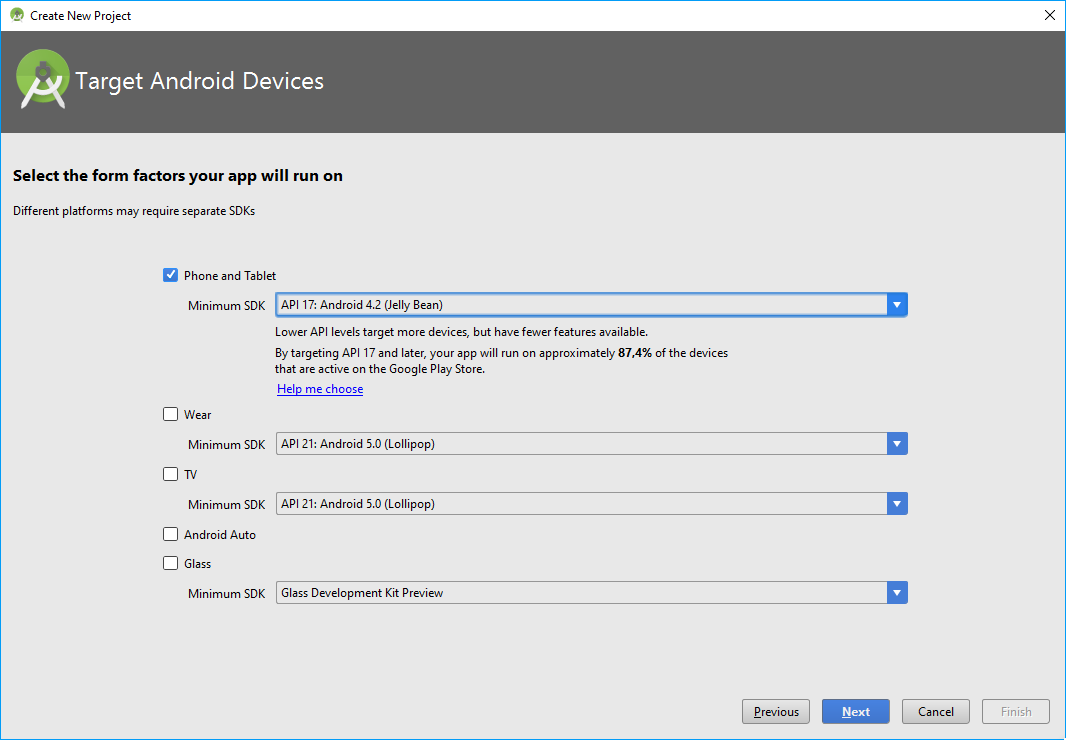
Щелкнем по опции Start a new Android Studio project (начать новый проект).



Вызовем диалоговое окно для выбора местоположения будущего проекта. Если хорошо работает сеть, лучше выбирать это место на сетевом диске. В именах папок, составляющих путь, не должно быть русских букв – только латинские буквы, цифры и нижние подчеркивания, имена не должны начинаться с цифр. То же самое относится к имени проекта.



Укажем имя приложения (здесь – StartApplication).

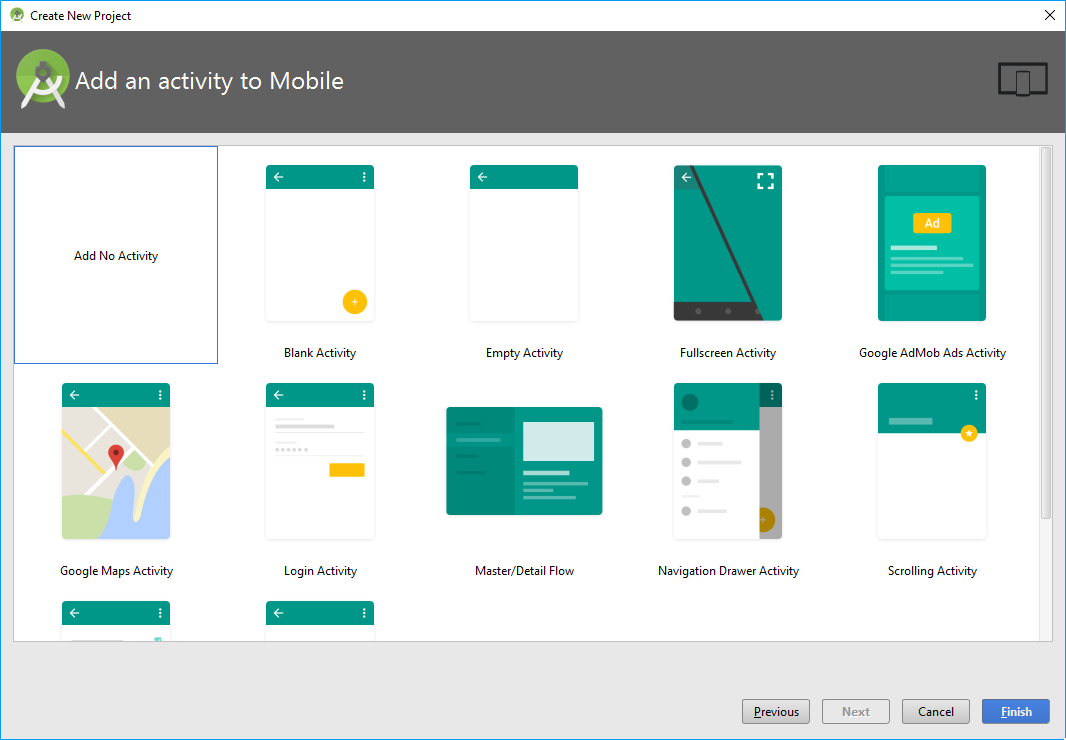


Выбираем SDK с учетом ресурсов большинства компьютеров ВЦ.

**SDK** (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *software development kit*) — комплект средств разработки, который позволяет специалистам по [программному обеспечению](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) создавать приложения…

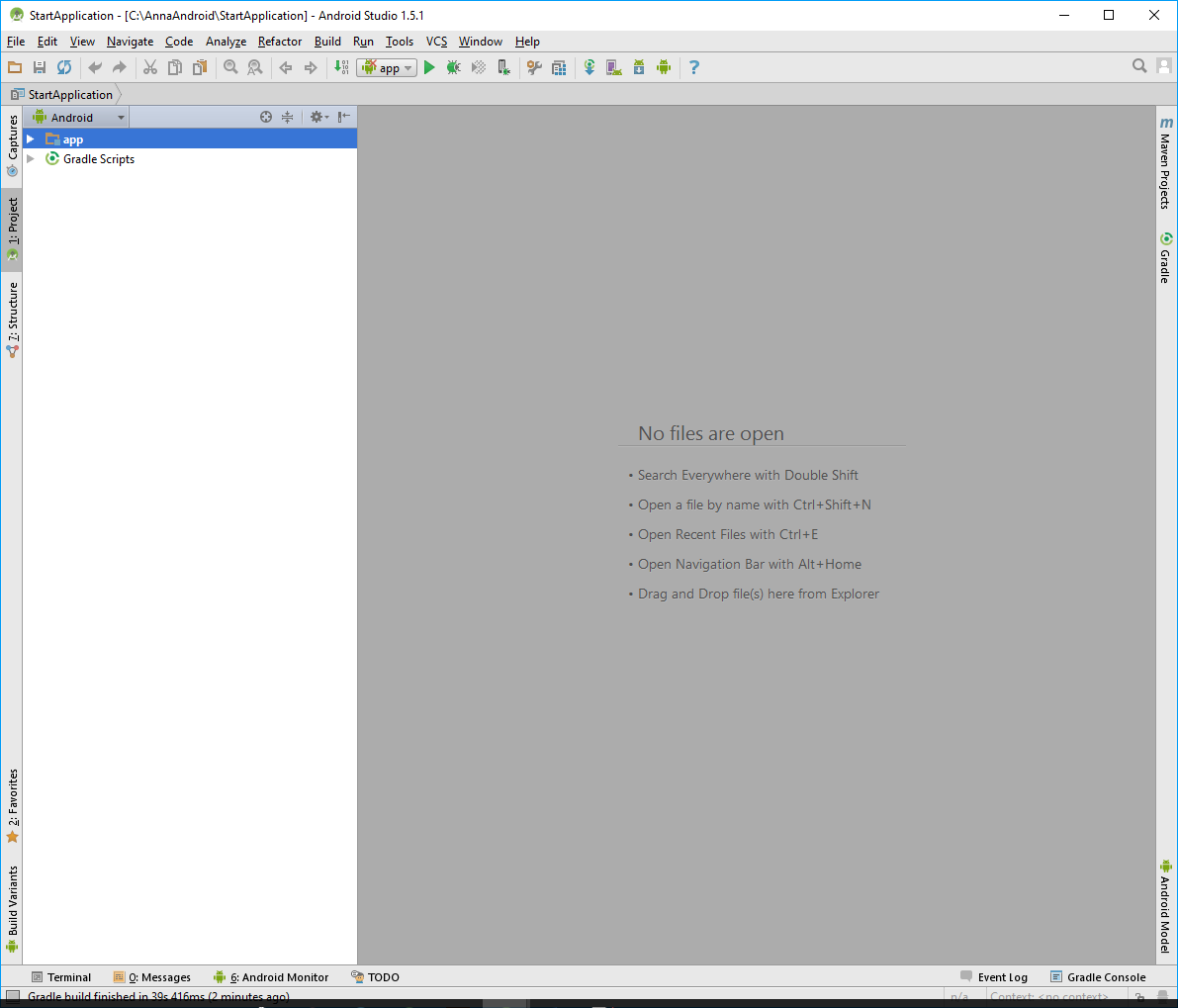
Поставщики *SDK* иногда подменяют слово «software» в словосочетании «software development kit» на более точное слово. Например, … [*Oracle*](https://ru.wikipedia.org/wiki/Oracle) называет свой инструментарий для разработки [*Java Development Kit*](https://ru.wikipedia.org/wiki/Java_Development_Kit) (*JDK*).

**https://ru.wikipedia.org/wiki/SDK**



Выбираем шаблон приложения без активности – активность будем создавать сами. Ближайший аналог термина ***активность*** – форма, с формами Вы, вероятно, работали в Delphi или в Visual Studio.

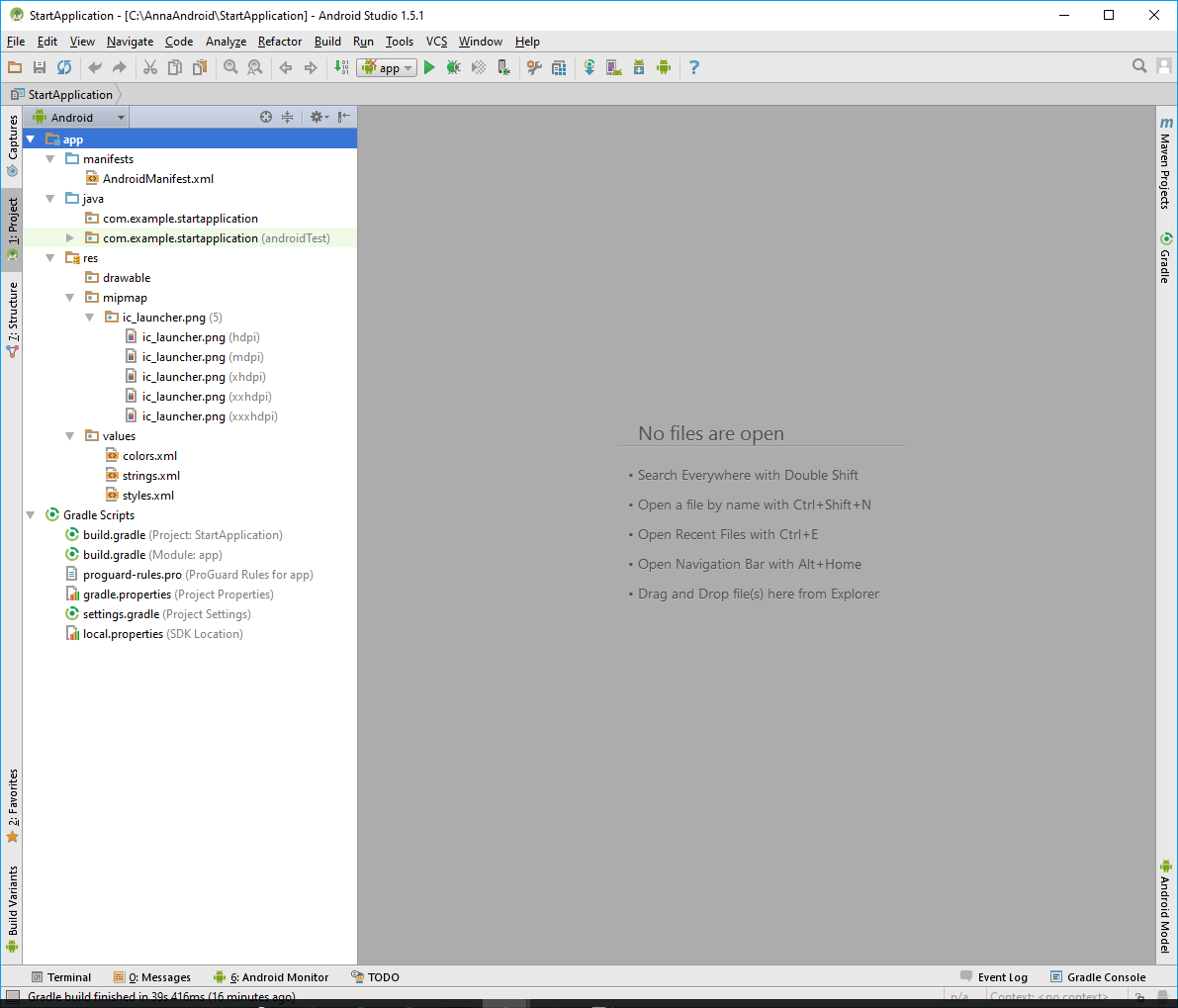
Далее через некоторое время должно появиться такое окно:



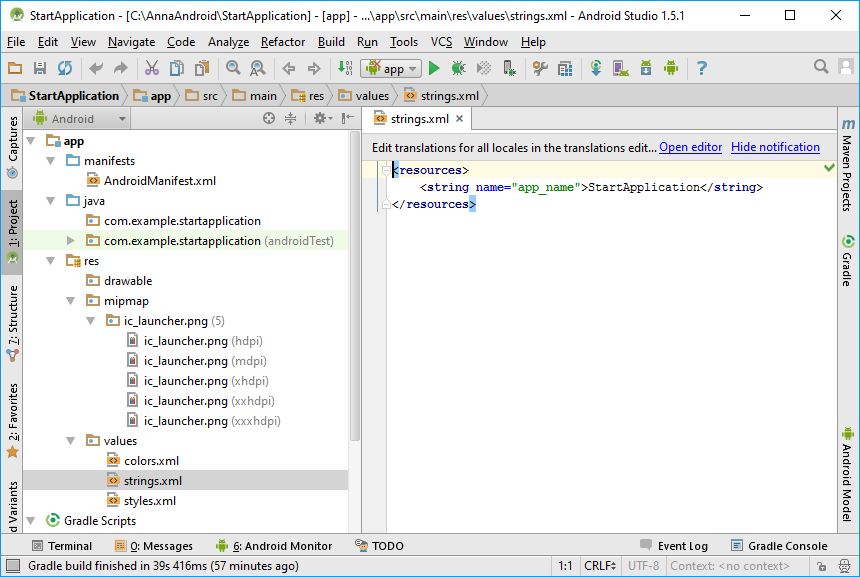
Требуется выбрать вкладку «Project» и из выпадающего списка – «Android», тогда мы увидим два основных элемента списка

* app (там содержится все, что относится к нашему приложению)
* Gradle Scripts (там содержится все, что относится к системе автоматической сборки)

Если мы раскроем списки, то увидим следующее:

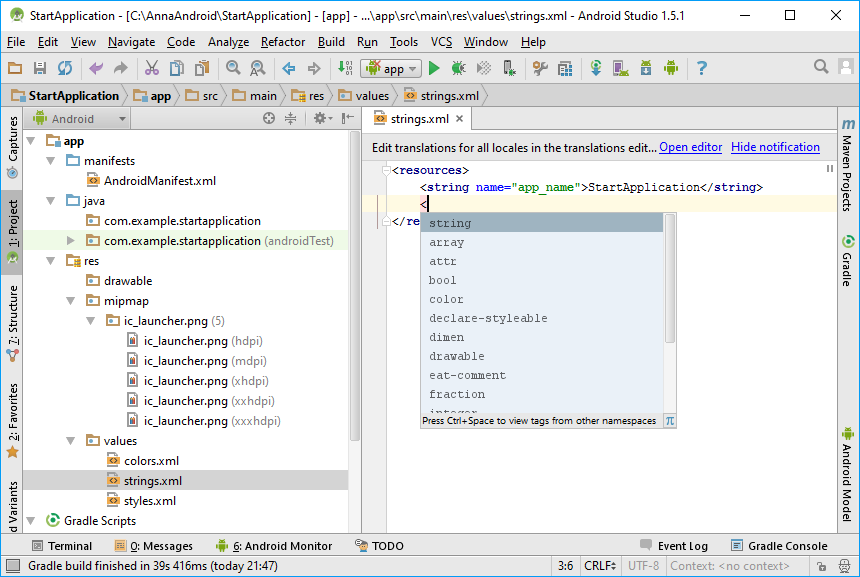


В папке ***res*** находятся ресурсы для нашего приложения, в папке ***res/values*** находятся ресурсы-значения. Откроем файл ***res/values/strings.xml*** со строковыми ресурсами-значениями (двойной щелчок по имени файла):



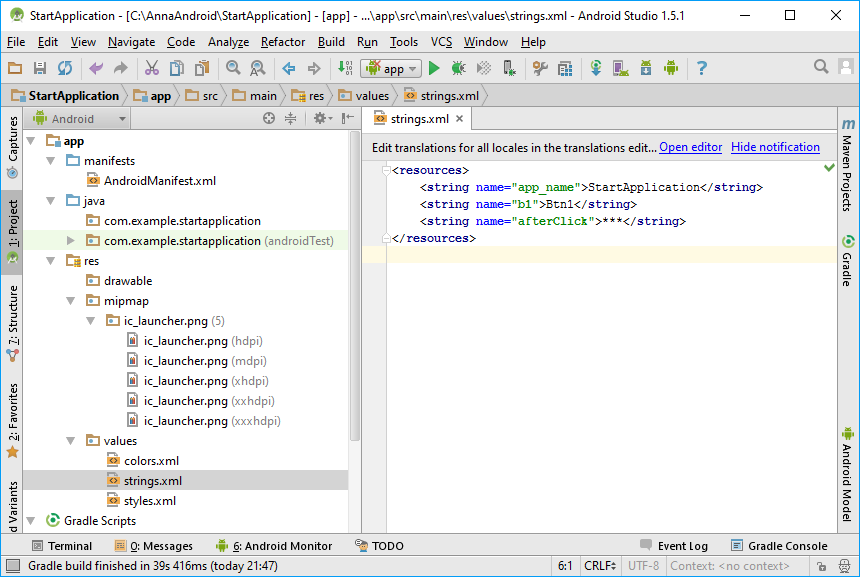
Откроется файл в формате xml, в котором внутри тега ***<resources>*** находится тег строки с названием нашего приложения. Здесь ***app\_name*** – имя строки (значение параметра ***name***), а ***StartApplication*** – сама строка.

Добавим сюда строки с надписями на кнопках сразу после запуска приложения и после нажатия на кнопку – Btn1, Btn2, Btn3, Btn4, \*\*\*. Обратите внимание на то, что имена строк должны быть уникальными. Android Studio помогает нам писать теги и код – стоит напечатать открывающую угловую скобку, как появляется подсказка, в которой мы можем выбрать нужную опцию с помощью стрелок и нажать ***Enter***. Мы работаем почти без использования мыши.

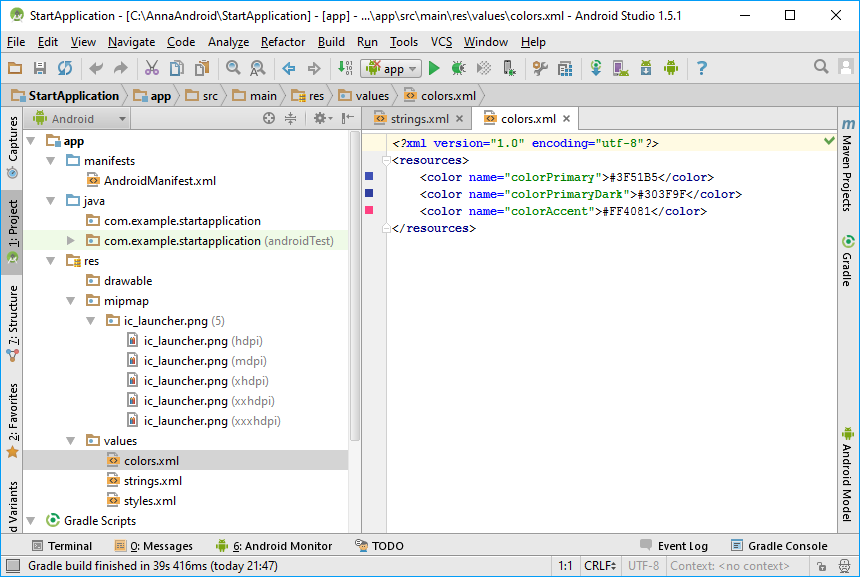


Здесь указаны 2 строки из 5-ти, остальные допишите сами.

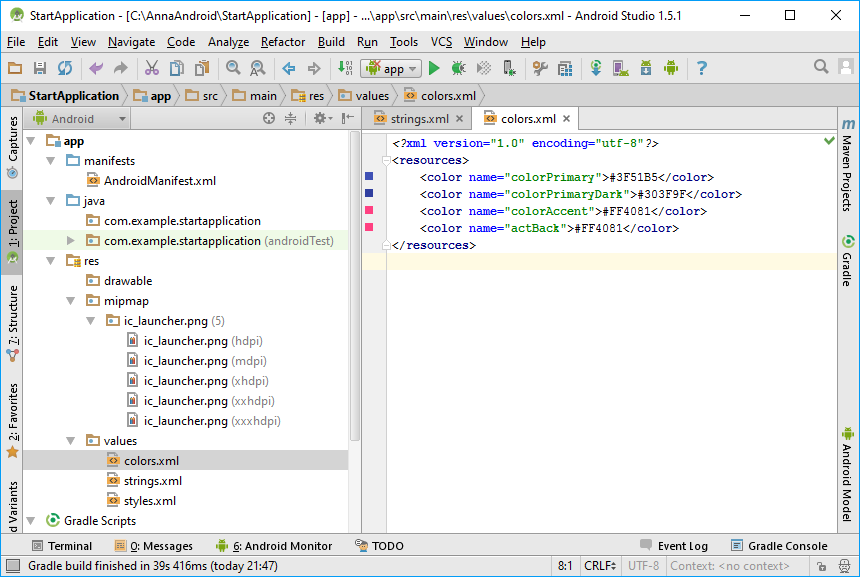
Имена строк должны удовлетворять требованиям для имен переменных – они состоят из латинских букв, цифр и нижних подчеркиваний, начинаются не с цифры. Содержимое строк может быть почти любым.



Теперь откроем файл ***colors.xml*** с ресурсами-цветами.



В файле находятся 3 стандартных цвета, добавим еще несколько цветов для фона самой активности (формы), текста и фона кнопок. Теги ***<color>*** со значениями цветов в формате RGB также, как и строки, находятся внутри тега ***<resources>***. В принципе, можно было бы разместить и цвета, и строки в одном файле, синтаксической ошибки это бы не вызвало, но так не принято делать из соображений удобства, принято группировать ресурсы разных видов по разным файлам. Создадим новый ресурс цвета, RGB-код цвета укажем пока любым:

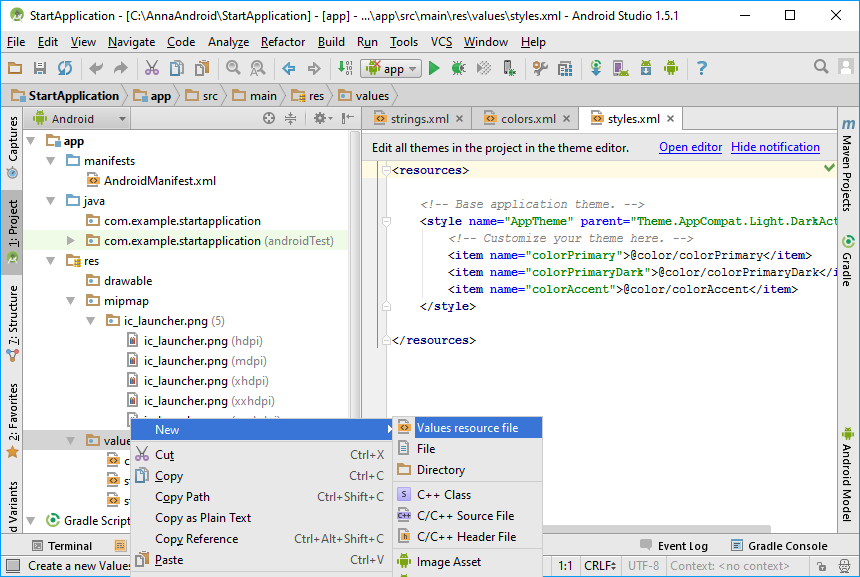


Слева от тега появится квадратик с образцом цвета. Щелкнув по этому квадратику, мы попадаем в палитру цветов, в которой мы можем выбрать желаемый цвет.



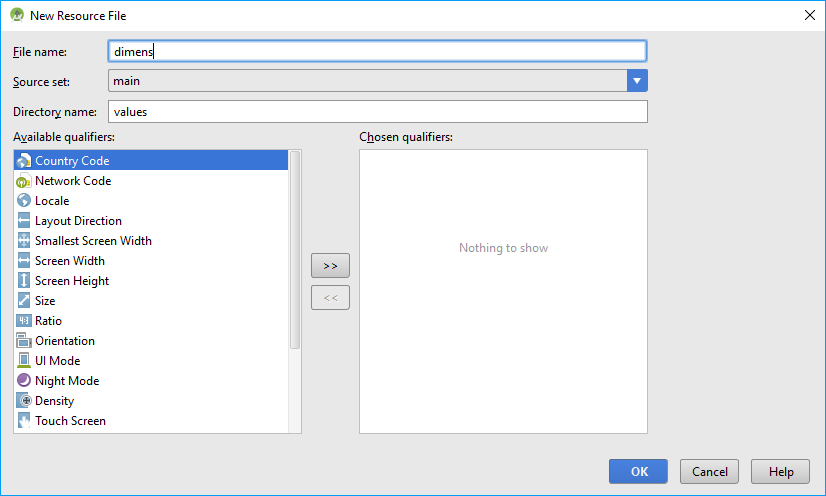
Всего у Вас должно быть добавлено 3 своих цвета – для фона активности, текста и фона кнопок. Имена цветов (значения параметра ***name*** тега ***color***), также, как и имена строк, должны быть уникальными.

Теперь давайте зададим числовые ресурсы (числовые константы) для внутренних и внешних отступов и размера шрифта. Числовым ресурсам соответствует тег ***<dimen>*** (от слова dimension - размерность). В автоматически созданных файлах для ресурсов значений (***colors, strings, styles***), мы не видим таких тегов, можно было бы поместить ***<dimen>*** в любой из этих файлов внутрь тега ***<resources>***, но давайте создадим для числовых ресурсов отдельный файл в папке ***values***, назовем его ***dimens***.

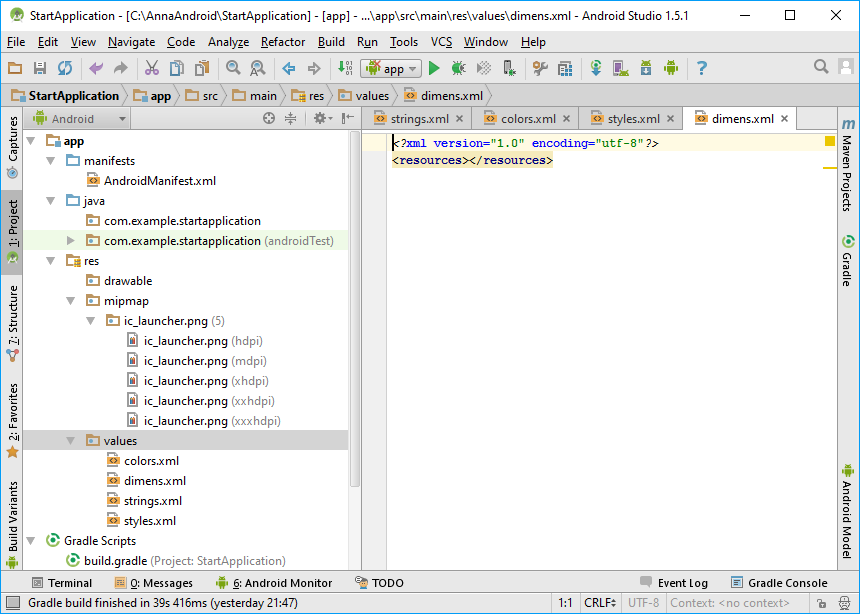


Подведите курсор мыши к папке values, нажмите правую клавишу мыши и из контекстного меню выберите указанные элементы.

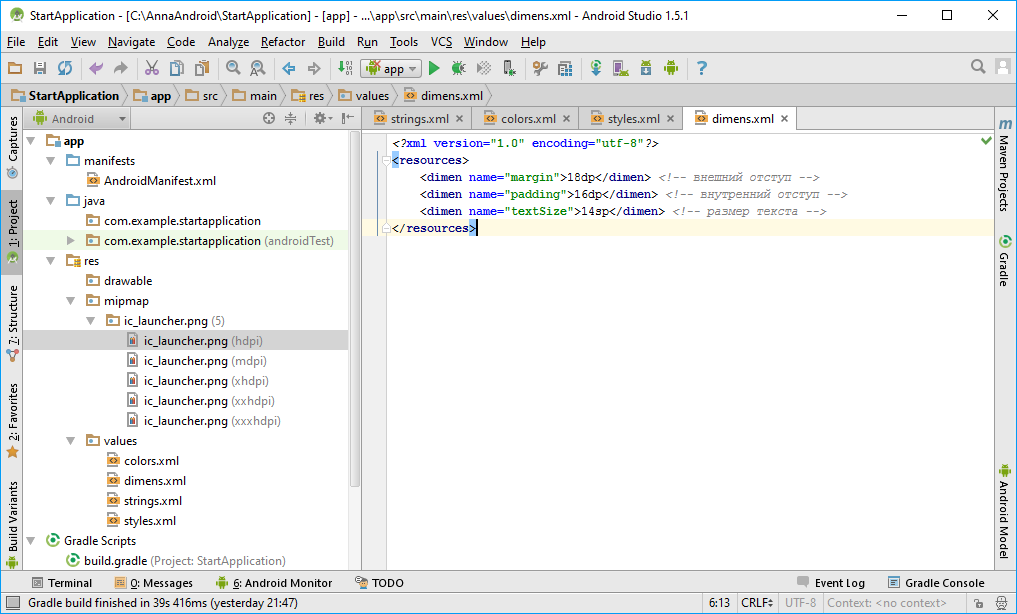
Откроется следующее окно:



Укажите маленькими латинскими буквами имя файла и нажмите OK.



Файл ***dimens.xml*** создался и открылся, в нем уже находятся открывающий и закрывающий теги ***<resources>***. Поместим в этот файл числовые константы – ресурсы:



<!-- так в файлах html и xml пишутся комментарии -->

Про ***dp*** и ***sp***:

Для указания ширины, высоты и отступов элементов разметки используются различные [единицы измерения](http://developer.android.com/guide/topics/resources/more-resources.html#Dimension) (ЕИ):

**dp** или **dip** - Density-independent Pixels. Абстрактная ЕИ, позволяющая приложениям выглядеть одинаково на различных экранах и разрешениях.

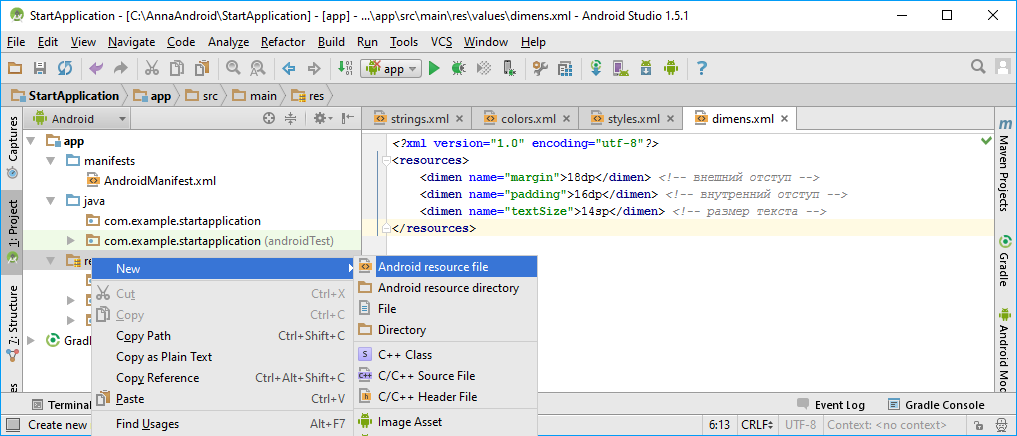
**sp** - Scale-independent Pixels. То же, что и dp, только используется для размеров шрифта в View элементах

**pt** - 1/72 дюйма, определяется по физическому размеру экрана. Эта ЕИ [из типографии](http://en.wikipedia.org/wiki/Point_%28typography%29).

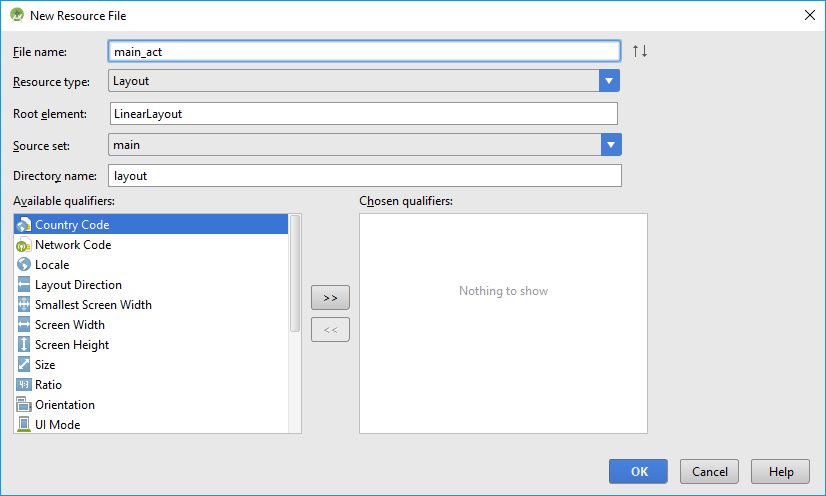
**px** – пиксел, не рекомендуется использовать т.к. на разных экранах приложение будет выглядеть по-разному.

<http://startandroid.ru/ru/11-pamjatka/40-edinitsy-izmerenija-chem-otlichaetsja-dp-dip-ot-px-screen-density.html>

Теперь давайте сконструируем саму активность (форму). Xml-файл с разметкой для активности должен находиться в папке res/layout. У нас пока нет папки layout (переводится как расположение). Один из способов создания этой папки и находящегося в нем xml-файла разметки: подведите курсор мыши к папке res, нажмите правую клавишу и из контекстного меню выберите указанные опции:

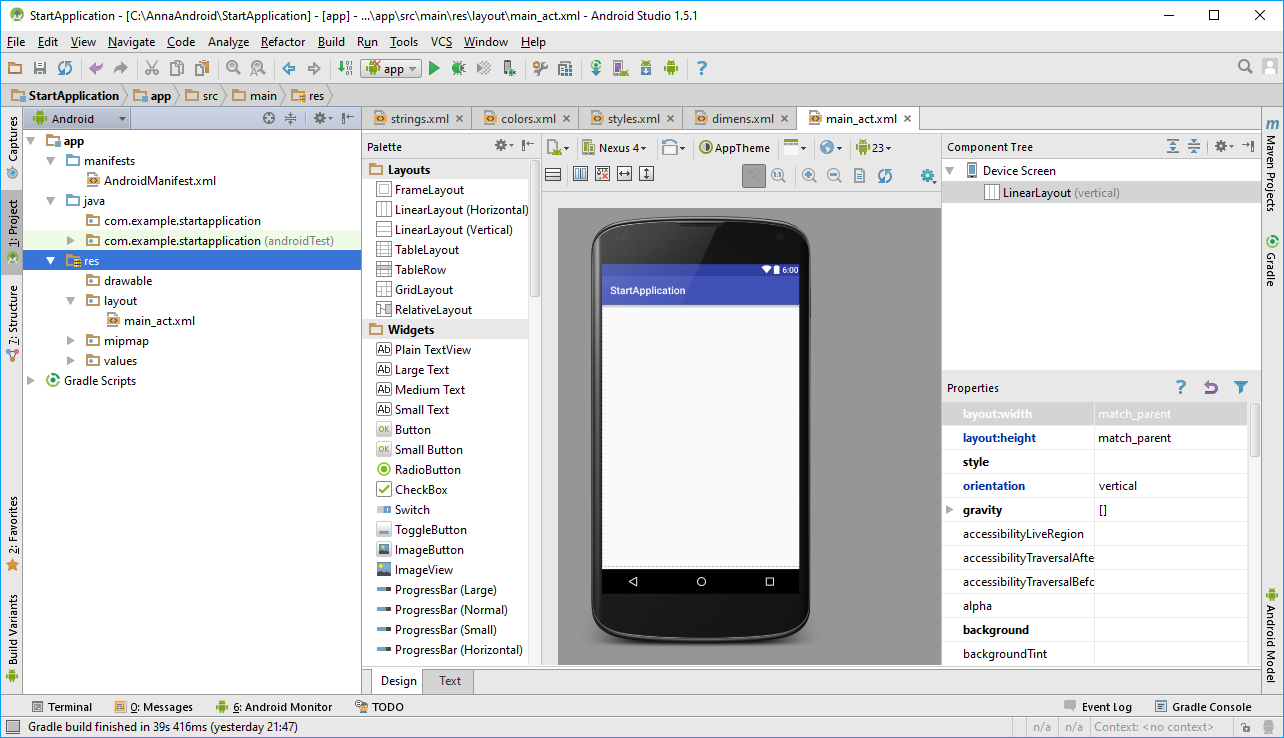


Откроется следующее окно:



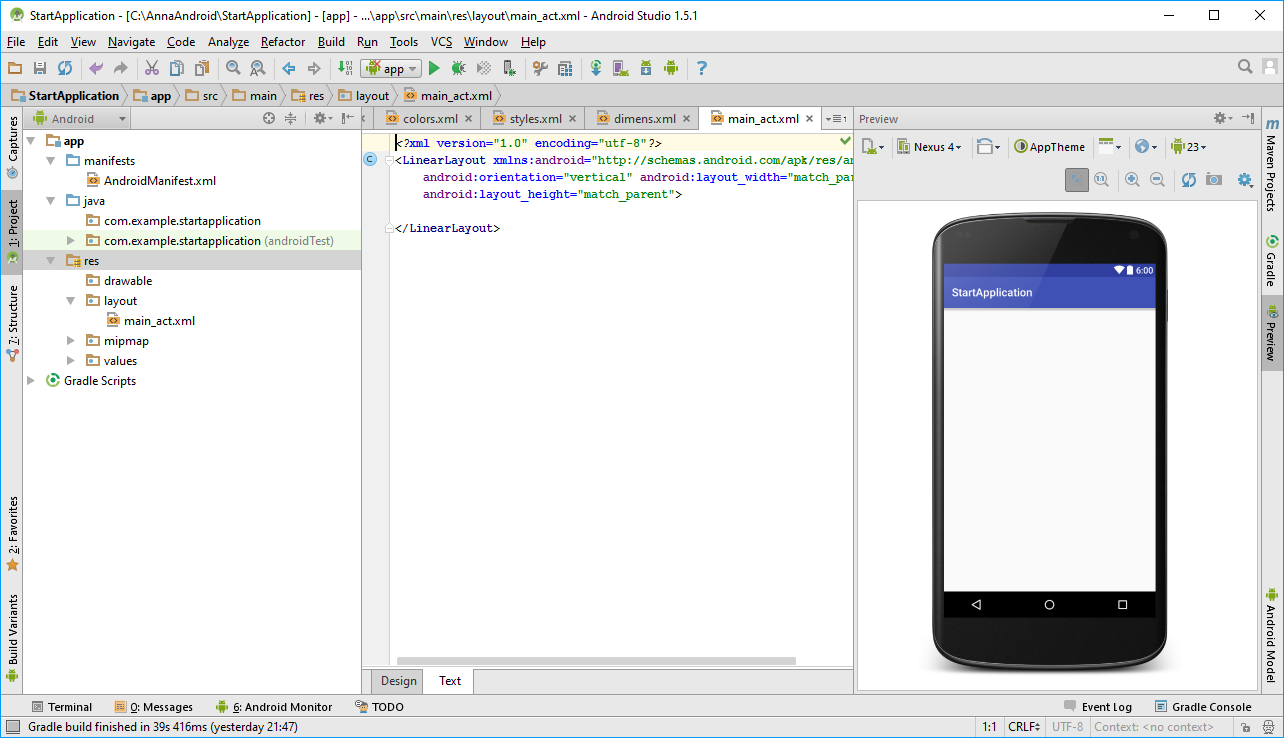
Имя файла пишем маленькими латинскими буквами, тип ресурса выбираем ***Layout***, корневой элемент – ***LinearLayout***.

Появится следующее окно:



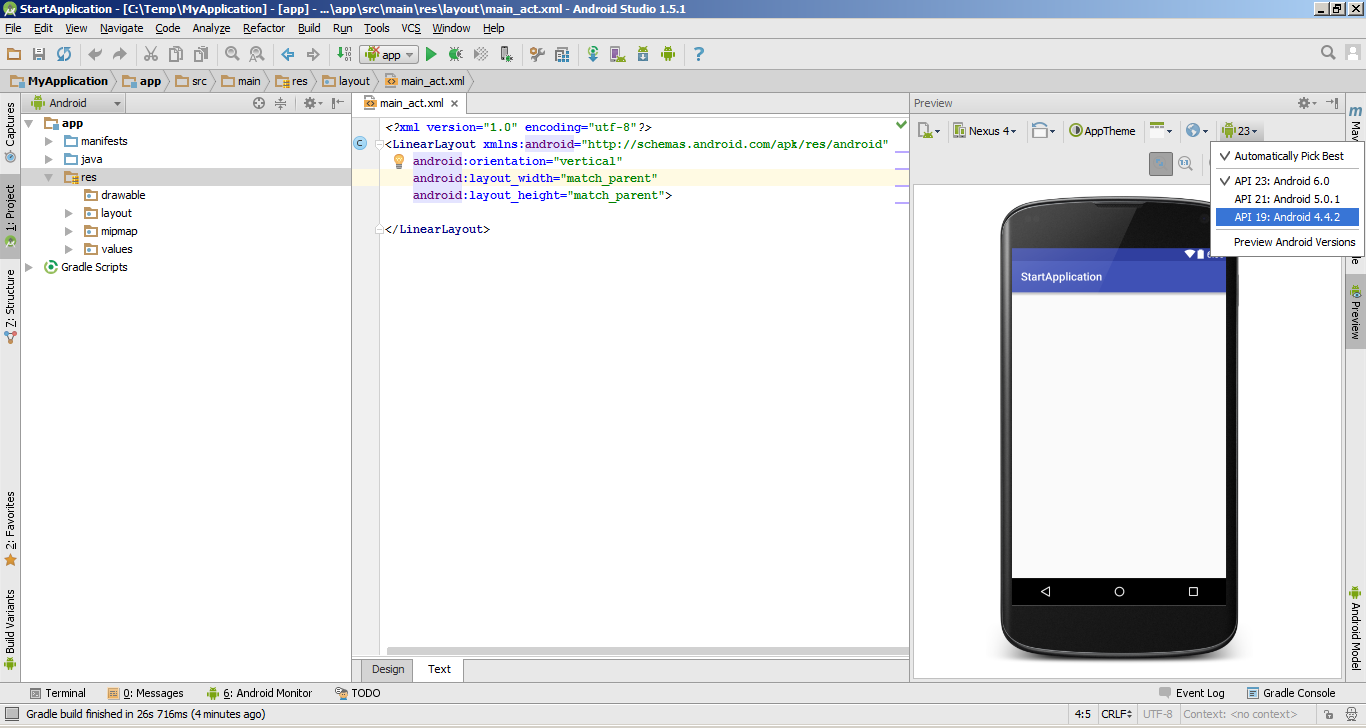
Здесь мы видим, что в папке ***res*** создалась подпапка ***layout***, можно ее раскрыть, щелкнув по значку треугольника слева, и увидеть содержимое – файл разметки ***xml***. Справа появится макет (это пока еще не эмулятор) мобильного устройства и откроется вкладка ***Design*** со списком компонентов/элементов управления, примерно как в ***Delphi*** или ***Visual Studio*** при работе с формой. Элементы управления можно помещать на макет, но мы откроем вкладку ***Text*** и будем создавать разметку с помощью xml-тегов.

Выберем вкладку ***Text*** и увидим следующее окно:

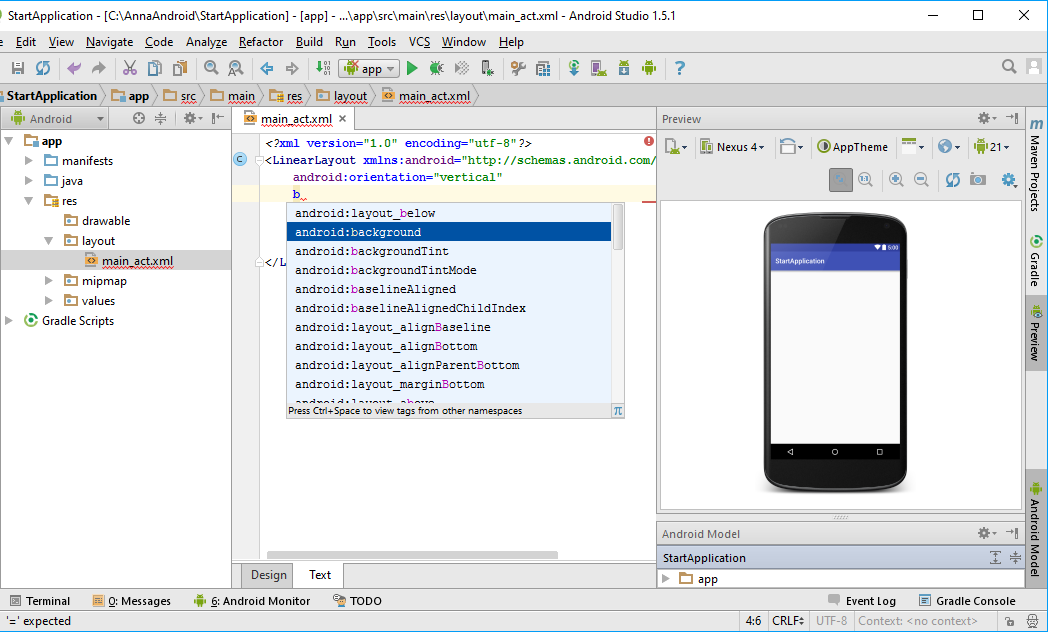


Здесь мы видим корневой и пока единственный элемент-тег ***LinearLayout***. Это тег-контейнер, все помещенные внутрь него элементы будут располагаться друг за другом по порядку в зависимости от ориентации (значения параметра тега ***orientation***). У всех элементов разметки есть обязательные параметры ***layout\_width*** и ***layout\_height*** – ширина и высота, в данном случае значение о@боих параметров – «***match\_parent***» (соответствие родительскому элементу), это значит, что данная активность будет занимать все окно.

Для того, чтобы при наших настройках лучше отображался макет, в выпадающем списке, выделенном красным прямоугольником, вместо версии «23» выберите версию «19» или «21».

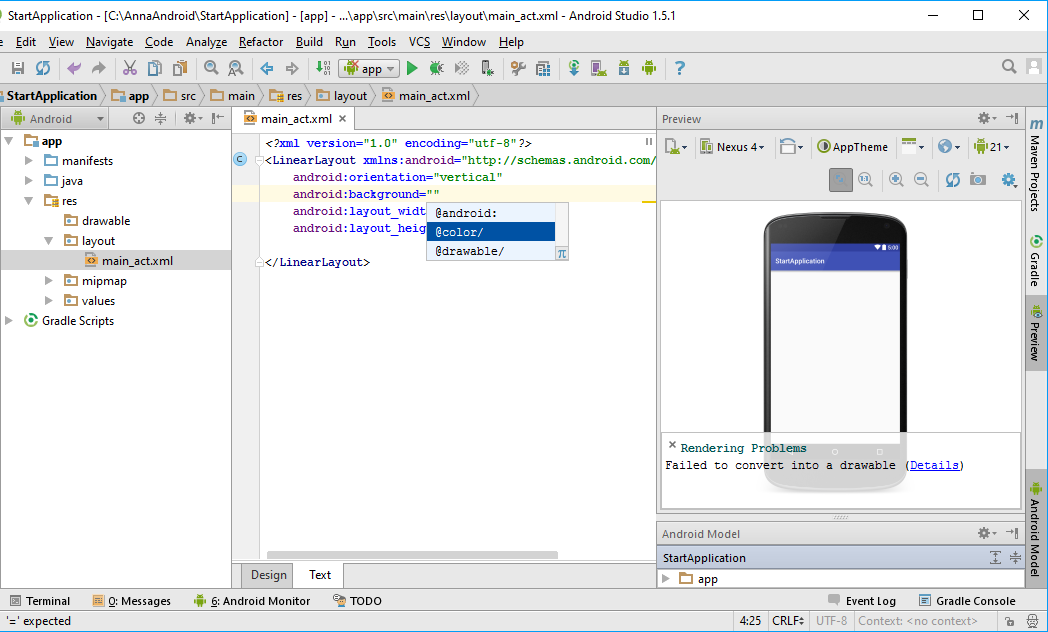


Давайте попробуем покрасить активность в какой-либо цвет, выбранный из ресурса цветов. Для этого укажем значение параметра «***background***» тега «***LinearLayout***». Android Studio опять проявляет свою дружественную природу – стоит набрать первую букву в названии параметра, как тут же появляется подсказка:

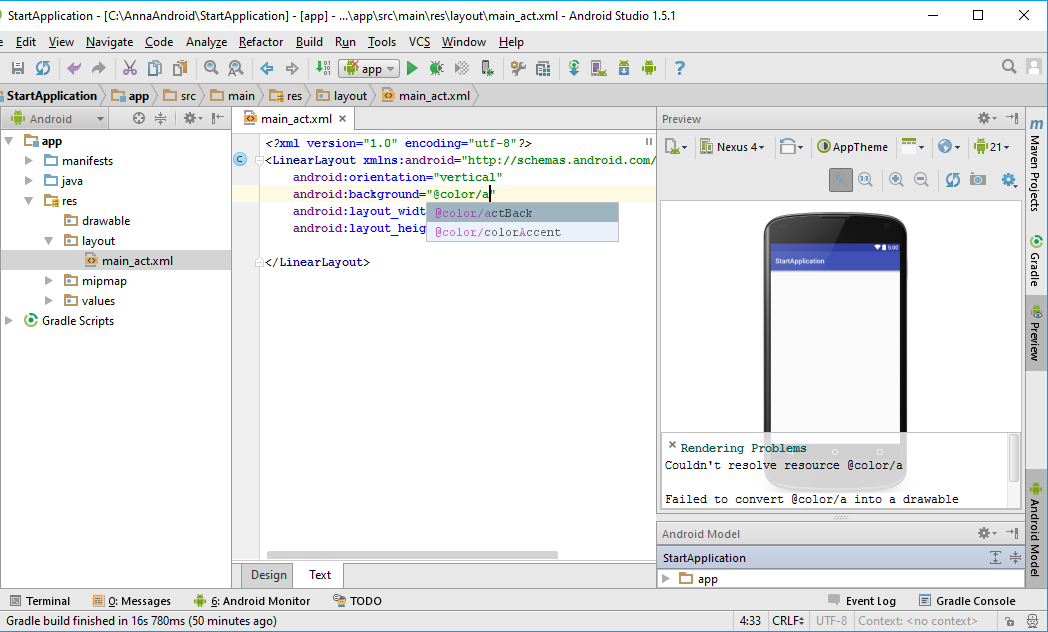


Далее появляется подсказка для выбора значения параметра, ***@*** - это признак ссылки на какой-либо ресурс, выберем ***@color*** – это ссылка на ресурс цвета, хранящийся в теге ***<color>***.

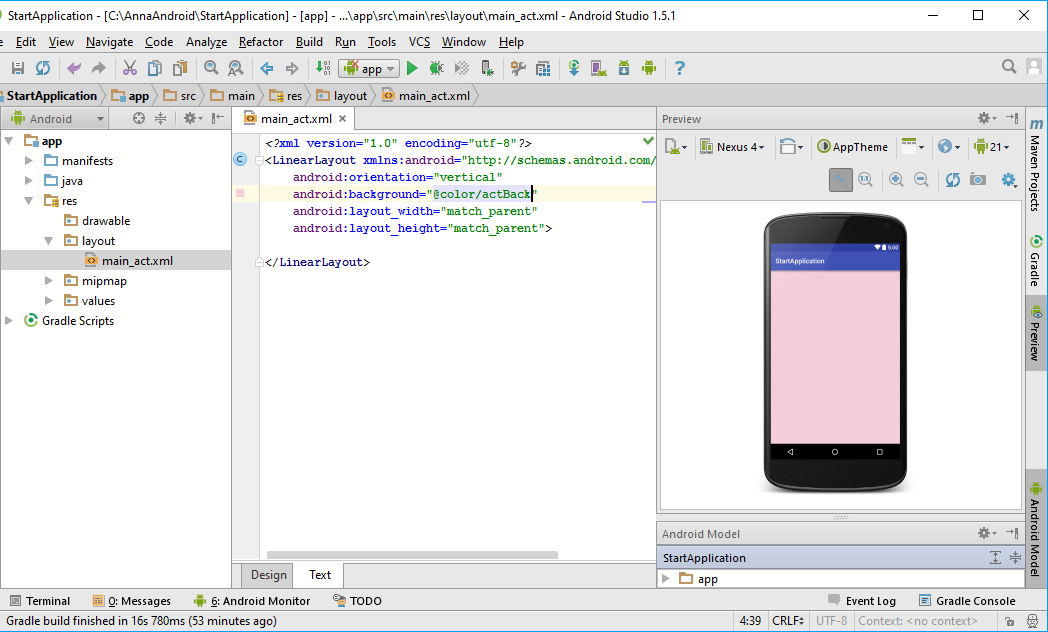
!!!Обратите внимание, что ***color*** – это название именно тега, а не файла.



Затем начинаем набирать имя цвета, предназначенного для фона активности:



Выбираем нужное имя и видим, что активность перекрасилась в выбранный цвет из ресурса:



Вспомним, что в файле ***colors.xml*** у нас был тег

***<color name="actBack">#f2cdda</color>***

Можно было бы указать цвет явно:

***android:background="#f2cdda"***

но в больших приложениях это неудобно и считается не очень хорошим тоном. Вспомните, например, почему в любом языке программирования рекомендуется использовать именованные константы вместо числовых.

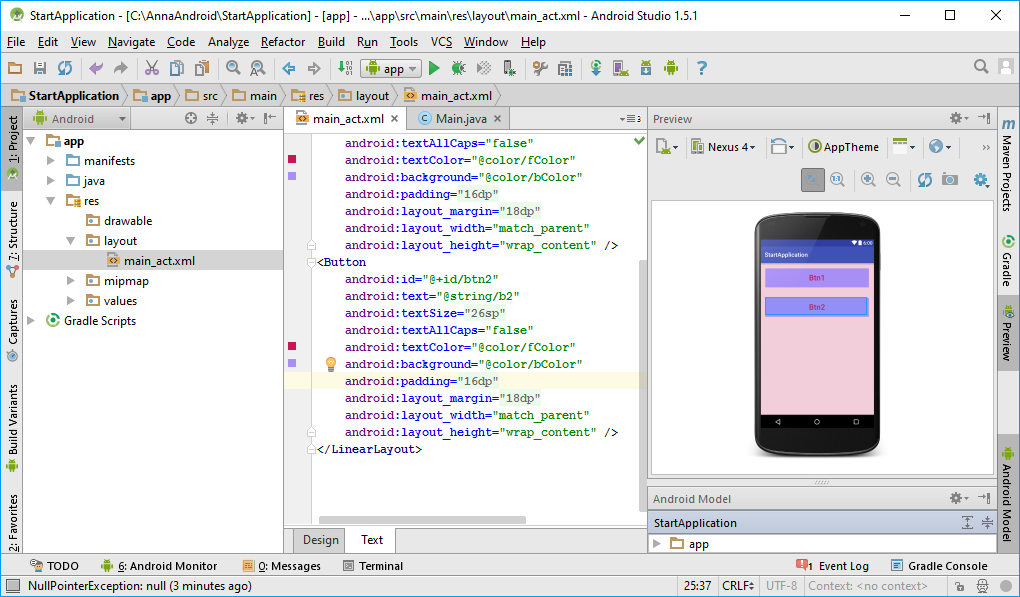
Поместим теперь на активность пару кнопок, указав для них текст на кнопке, внешние и внутренние отступы, цвет текста и фона и размер текста (делаем ссылки на строковые, числовые и цветовые ресурсы). Укажем ширину кнопок «***match\_parent***» - по ширине кнопке займут все окно, а высоту кнопок – «***wrap\_content***» - высота будет подбираться автоматически под размер содержимого. В Delphi или Visual Studio «***wrap\_content***» соответствует «***AutoSize***». Параметр

***android:textAllCaps="false"***

используется для того, чтобы не все буквы на кнопках были заглавными.

Конечно, для того, чтобы сделать одинаковые по внешнему виду кнопки, лучше использовать не большое количество одинаковых параметров, а стили, но со стилями мы познакомимся позже.

Параметр ***id*** используется для однозначной идентификации элемента для последующего обращения к этому элементу в коде программы. Все ***id*** в пределах одного файла разметки должны быть уникальными.

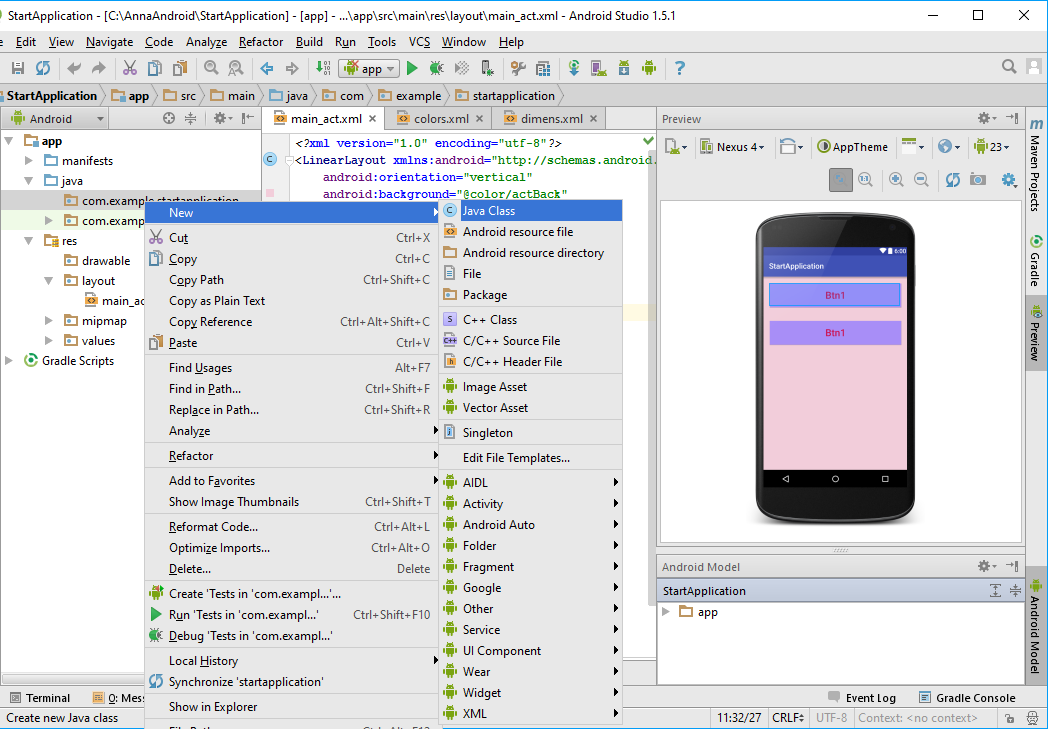


Добавьте на активность еще пару кнопок самостоятельно.

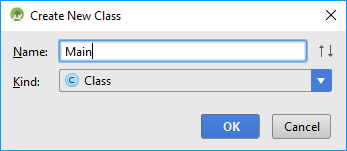
Теперь перейдем к написанию кода. Программировать мы будем на языке ***Java*** (не путать с JavaScript). Синтаксис языка ***Java*** в первом приближении похож на синтаксис всех С-подобных языков.

Каждой активности приложения соответствует свой класс. Давайте создадим класс, соответствующий нашей единственной активности.

Раскройте список ***java***, подведите курсор к первому элементу списка, нажмите правую клавишу мыши и из контекстного меню выберите указанные опции:

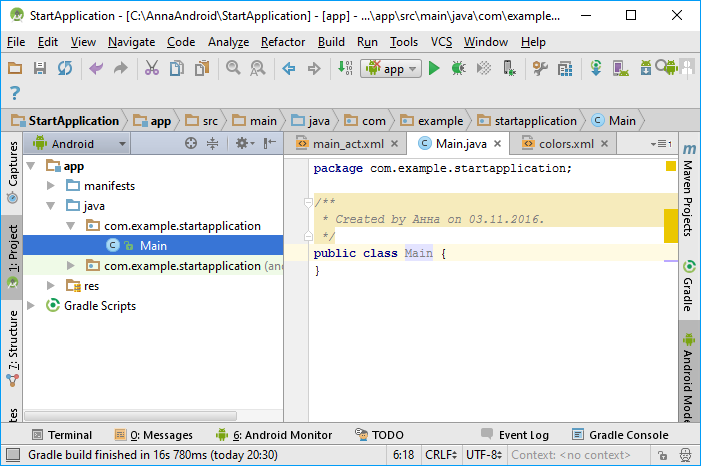


Вы увидите следующее окно:

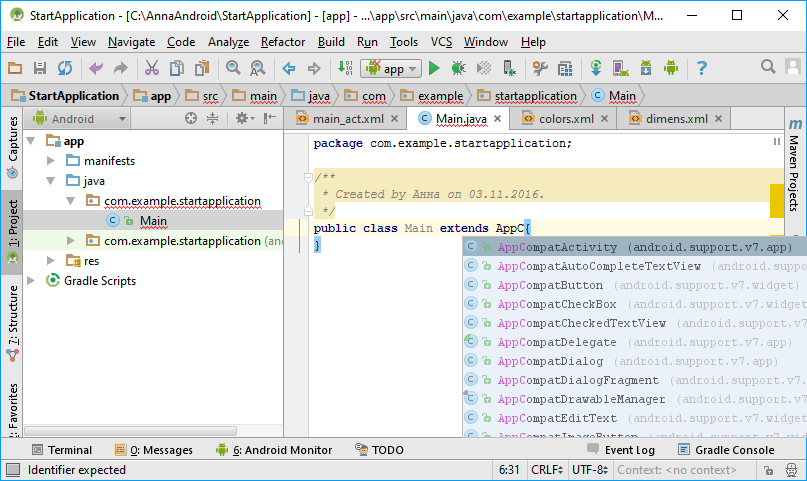


Укажите имя класса (в Java принято имена классов начинать с заглавной буквы).

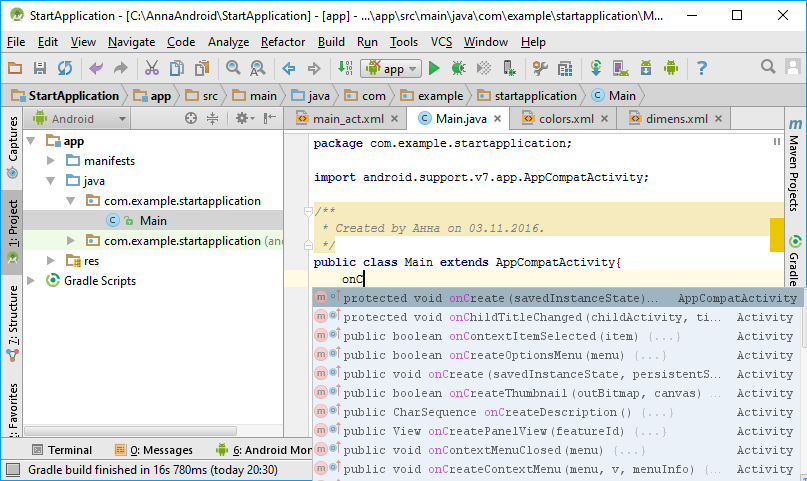
Далее откроется окно с заготовкой для нашего класса:



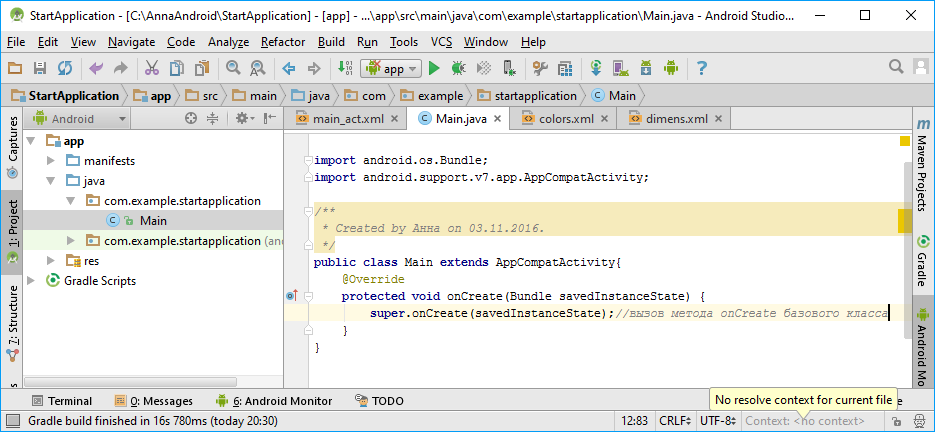
Любой класс, соответствующий активности, должен быть наследником от базового класса-активности. Есть несколько таких базовых классов, выберем ***AppCompatActivity*** и сделаем наш класс ***Main*** наследником от него. В ***Java*** ключевое слово ***extends*** означает, что класс (в нашем случае - ***Main***) является расширением, или наследником от другого класса (в нашем случае от ***AppCompatActivity***). При написании кода тоже постоянно появляются подсказки, стоит только набрать одну или несколько начальных букв.



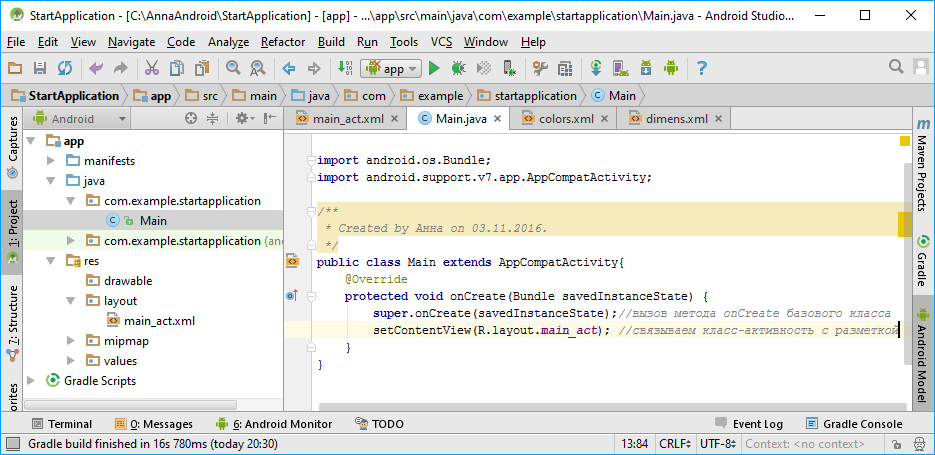
Далее нам потребуется перегрузить метод ***onCreate*** (!!! Выбираем метод ***onCreate*** с одним параметром).



Android создаст заготовку для метода ***onCreate***, который является обработчиком одноименного события, возникающего при создании активности.

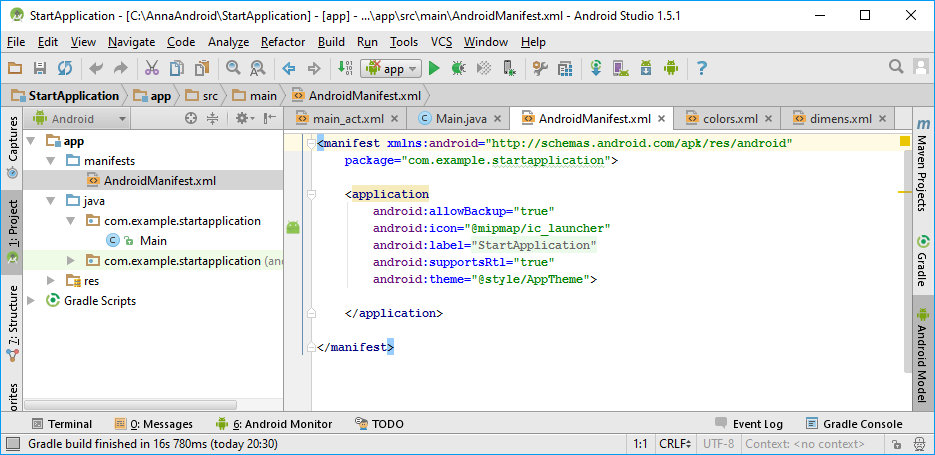


Теперь нам надо связать наш класс-активность с созданной для этой активности разметкой. Мы делаем это с помощью метода ***setContentView***, в качестве параметра указываем ссылку на xml-файл с разметкой. ***R*** перед именем файла – это класс, работающий с ресурсами.

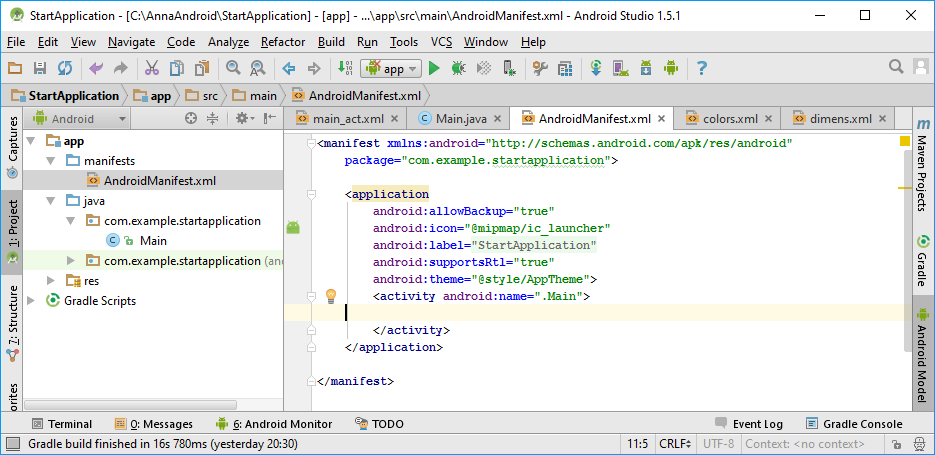


Метод ***setContentView*** создает из тегов-элементов разметки дерево программных объектов.

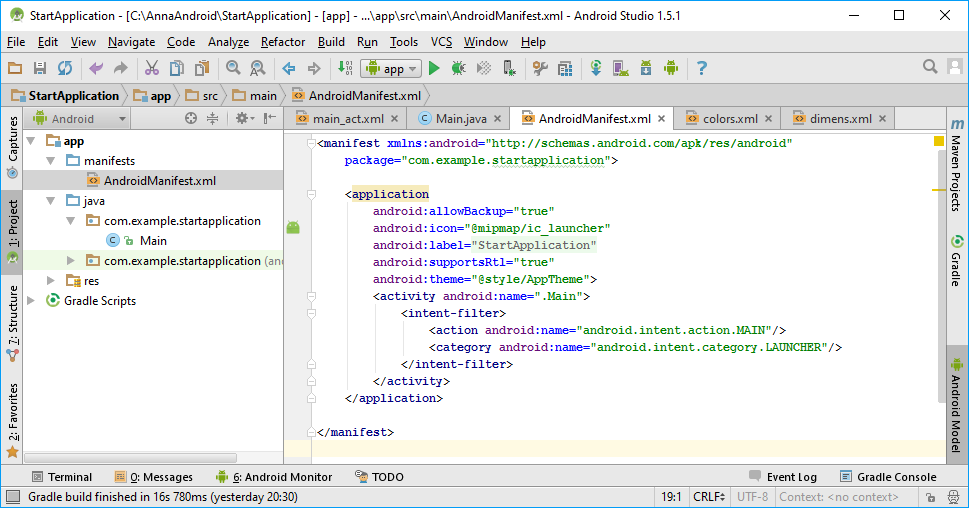
Далее нам надо представить нашу активность операционной системе – это делается в манифесте. Пока мы этого не сделаем, у значка-приложения будет находиться красный крестик и мы не сможем это приложение запустить. В файле манифеста уже есть заготовка:



Добавляем туда сведения о нашей активности:

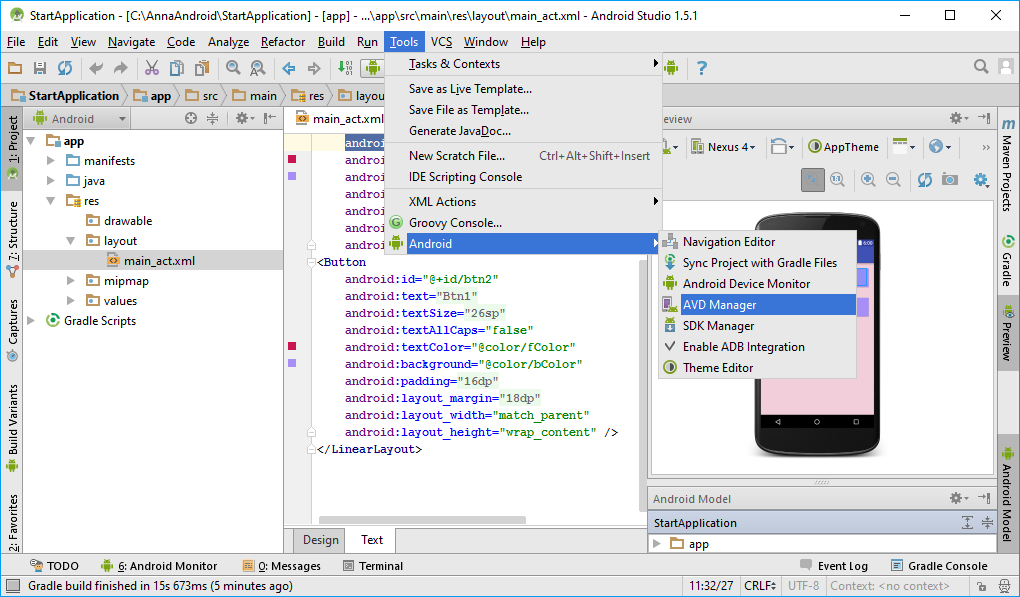


Далее надо указать операционной системе, что эта активность – главная, т.е. при запуске приложения именно она должна появиться на экране (по аналогии с главной формой для приложения в Delphi или Visual Studio). Это делается следующим образом:

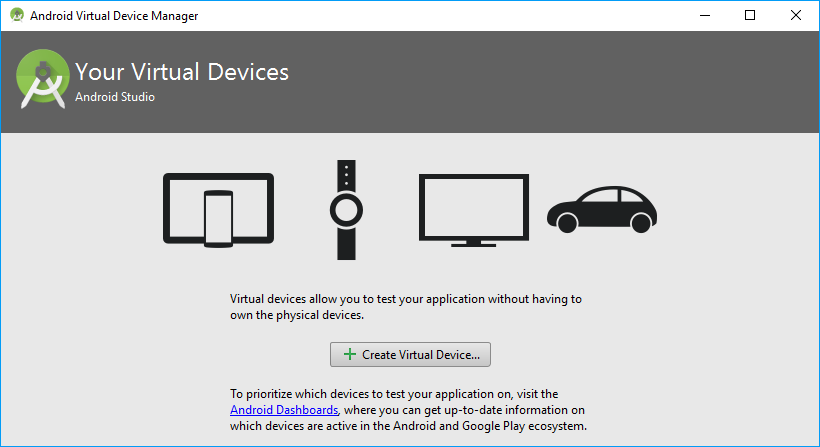


Теперь крестик исчез, и мы можем запустить наше приложение на эмуляторе (а у кого есть реальное устройство, можно запустить и на нем), хотя в нашем приложении пока нет обработчиков событий и оно не функционально.

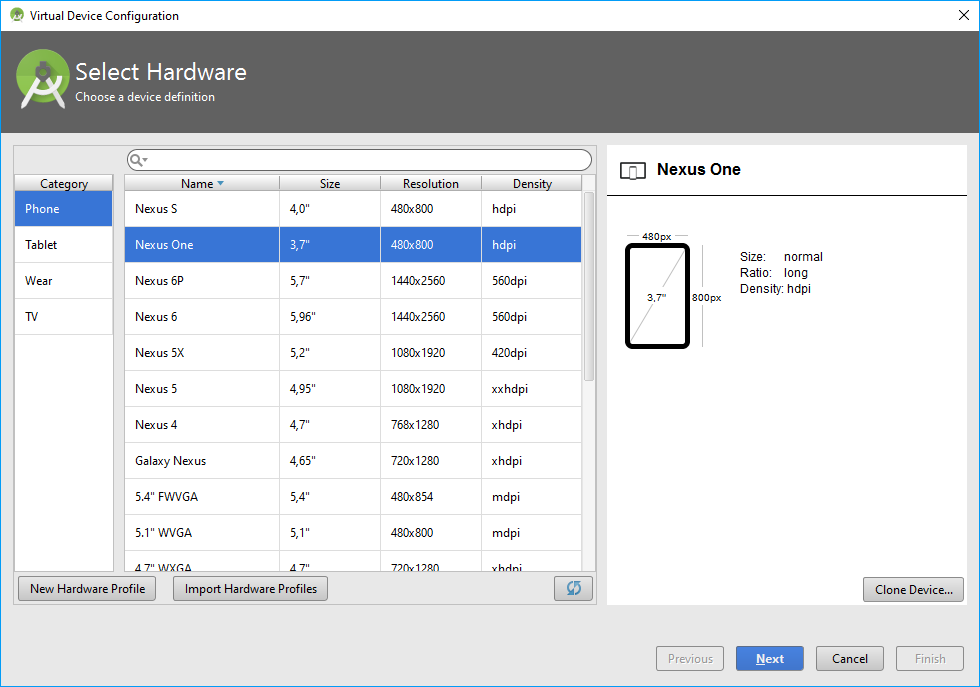
Можно создать несколько виртуальных устройств – эмуляторов, а потом выбрать одно из них для запуска приложения. Мы создадим одно устройство. Устройства можно создавать разными способами, используем следующий способ:



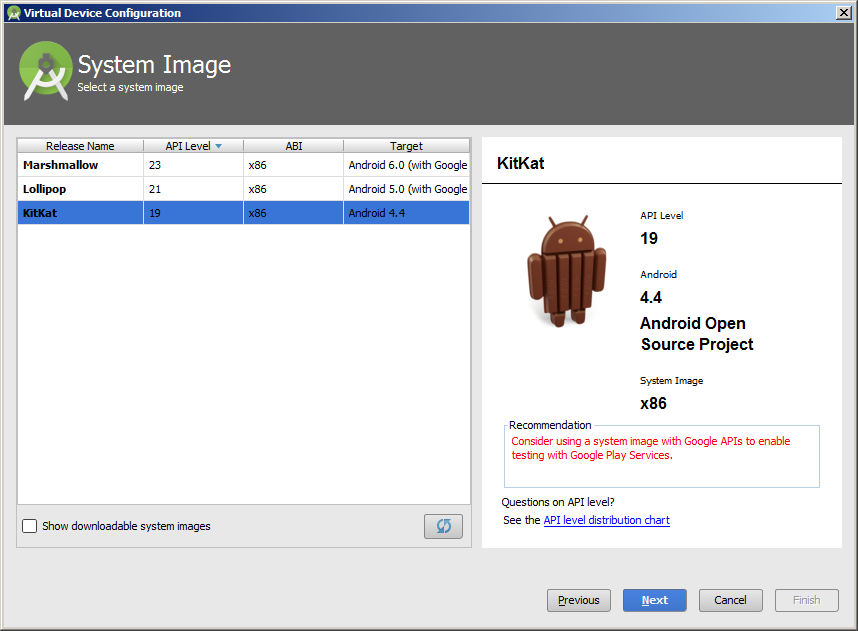
Выберем опцию меню AVD Manager (Android Virtual Device – виртуальное устройство Android).

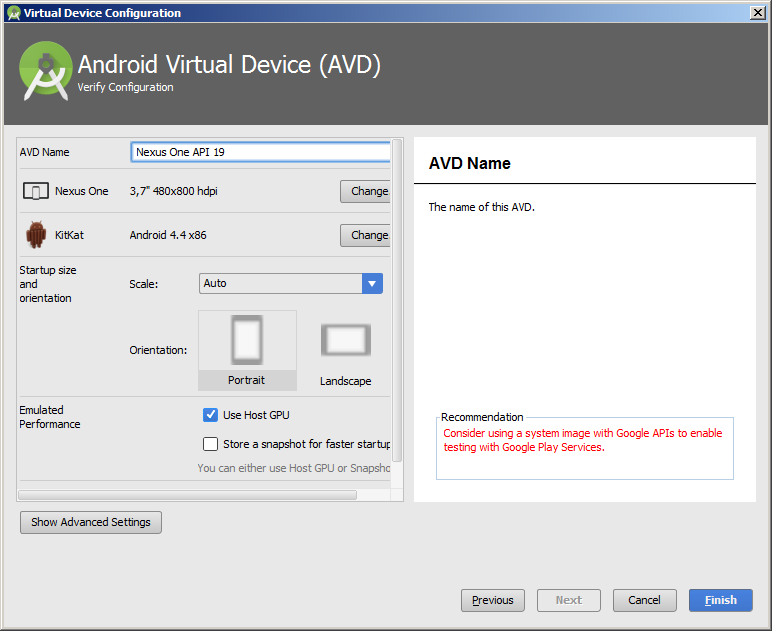


Если у нас пока еще нет созданных виртуальных устройств, мы увидим такое окно, в котором следует нажать кнопку «***Create Virtual Device***» (создать виртуальное устройство).

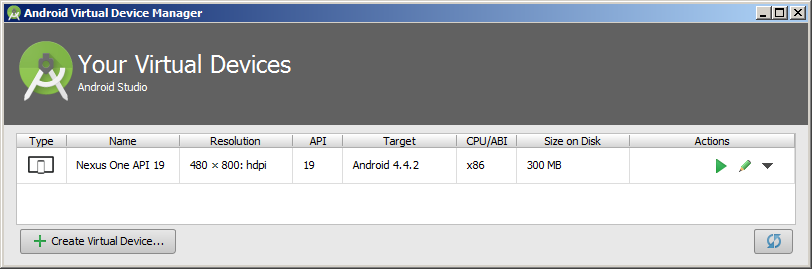


Выбираем эмулятор в соответствии с возможностями компьютеров ВЦ (конечно, если Вы работаете на своем ноутбуке, у Вас будет, скорее всего, более широкий выбор).



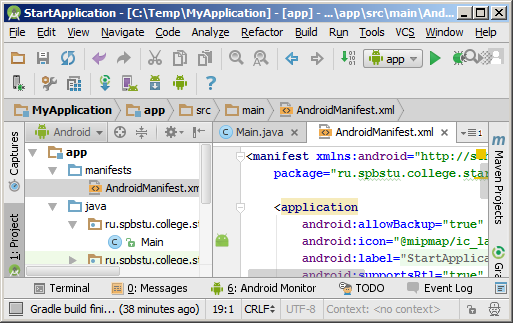


Созданное виртуальное устройство появится в списке устройств.

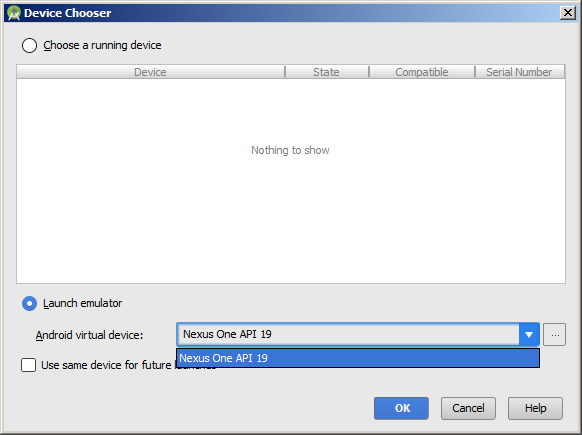


Можно создать другие виртуальные устройства, можно запустить наше устройство на эмуляторе прямо из этого окошка, но мы свернем это окошко и запустим устройство из другого окна, как это будет показано далее.

Начнем запускать наше приложение:

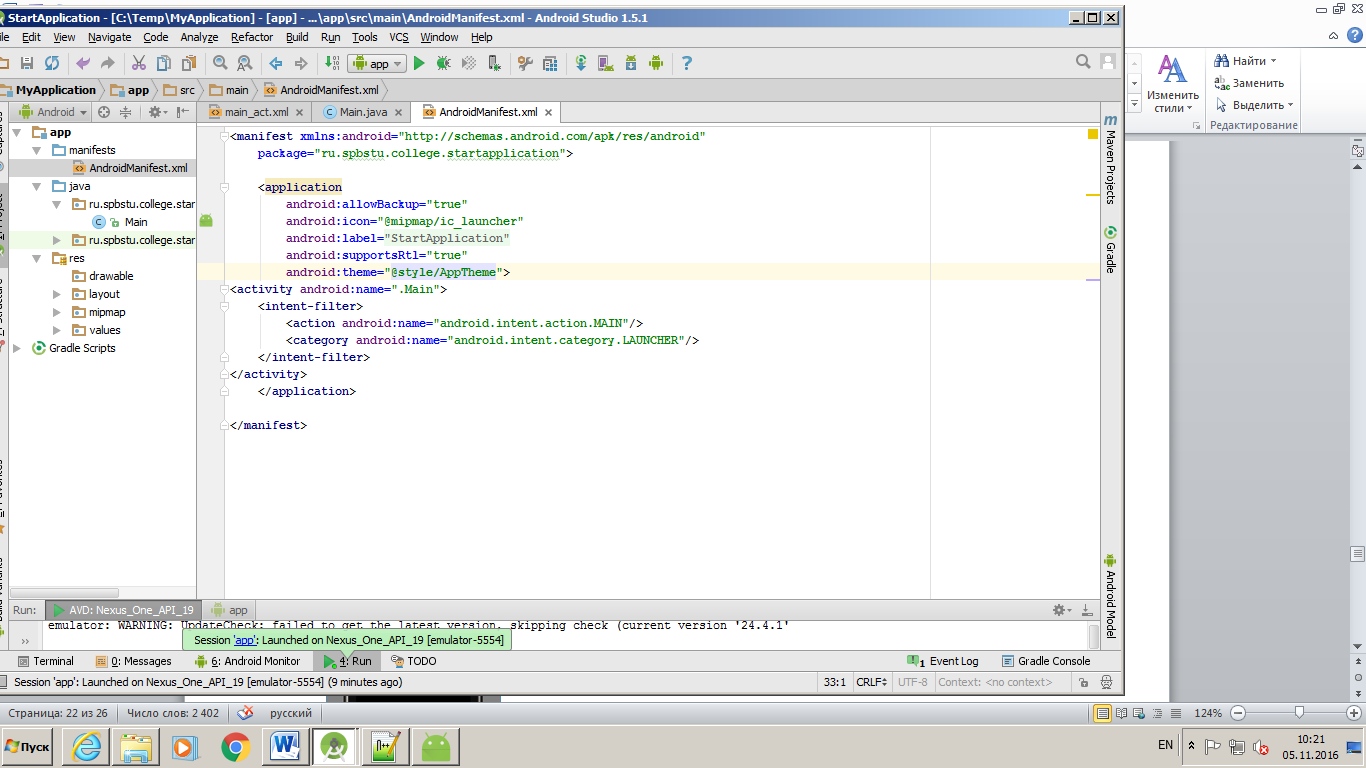


Поскольку у нас пока эмулятор не запущен, мы его запустим, выбрав предварительно созданное устройство, с которым мы хотим сейчас работать (у нас такое устройство одно).

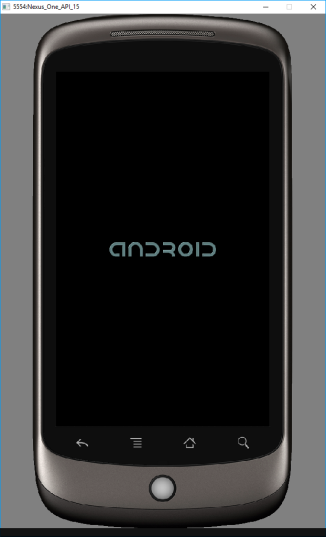


Можно было бы создать устройство прямо из этого окна.

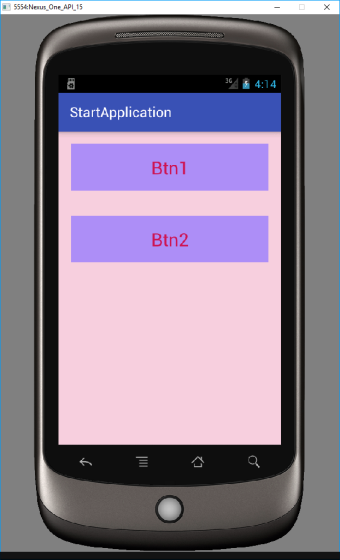
Через некоторое время появится эмулятор с запущенным на нем устройством, ему соответствует иконка:



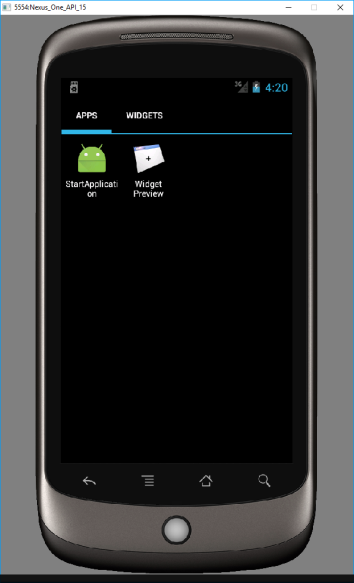
Начинает запускаться эмулятор:



Затем запустится приложение:



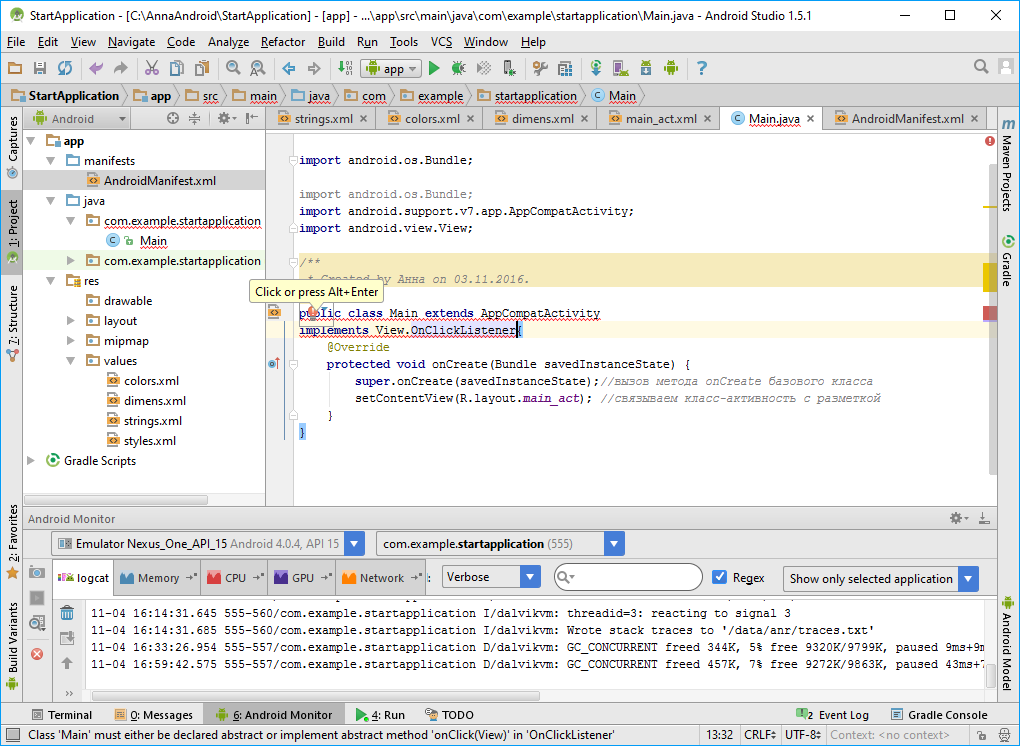
Из приложения можно выйти,



а потом опять его запустить.

Пока у нас не работают кнопки, т.к. не написаны обработчики для события «***щелчок по кнопке***». Можно по-разному привязывать обработчик к событию элемента управления, в первой работе мы будем использовать нижеуказанный способ:

* обязуем наш класс ***Main*** реализовать интерфейс ***View.OnClickListener.*** Вспомним, что интерфейс – это специального вида класс (это – одно из значений слова «интерфейс», которое нас сейчас интересует), который содержит только заголовки методов, не содержит тела методов, предполагается, что тела методов интерфейса будут написаны в классах-наследниках интерфейса. Для обозначения того, что класс реализует интерфейс, используется ключевое слово ***implements*** (***extends*** используется, если один класс является расширением-наследником другого класса – НЕ интерфейса)
* т.к. наш класс Main реализует интерфейс ***View.OnClickListener,*** в нашем классе должен быть реализован единственный метод этого интерфейса – ***onClick,*** т.е. для этого метода должно быть написано тело – это и будет обработчик для кнопок
* далее требуется создать программные объекты, соответствующие кнопкам ***Button*** разметки, и привязать их к обработчику ***onClick***

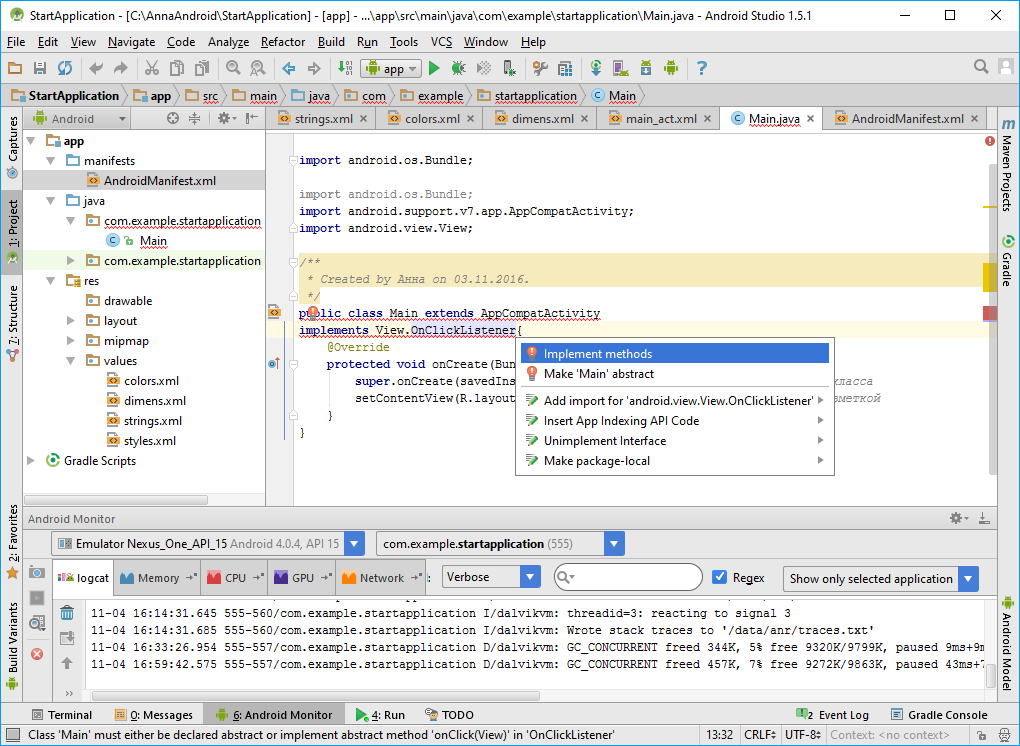


После того, как мы запишем код:

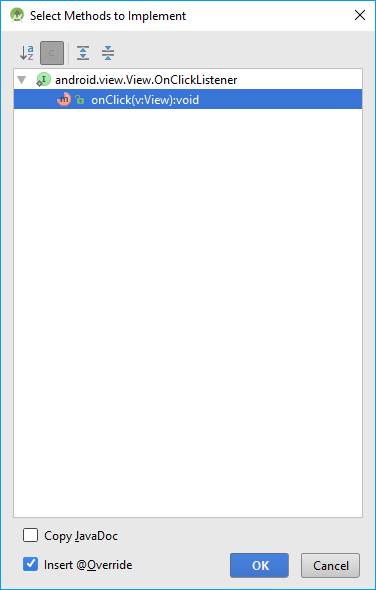
***implements View.onClickListener***

Android подчеркнет его красным и выведет рядом красную лампочку, сигнализирующую об ошибке. Внизу мы увидим сообщение об ошибке.

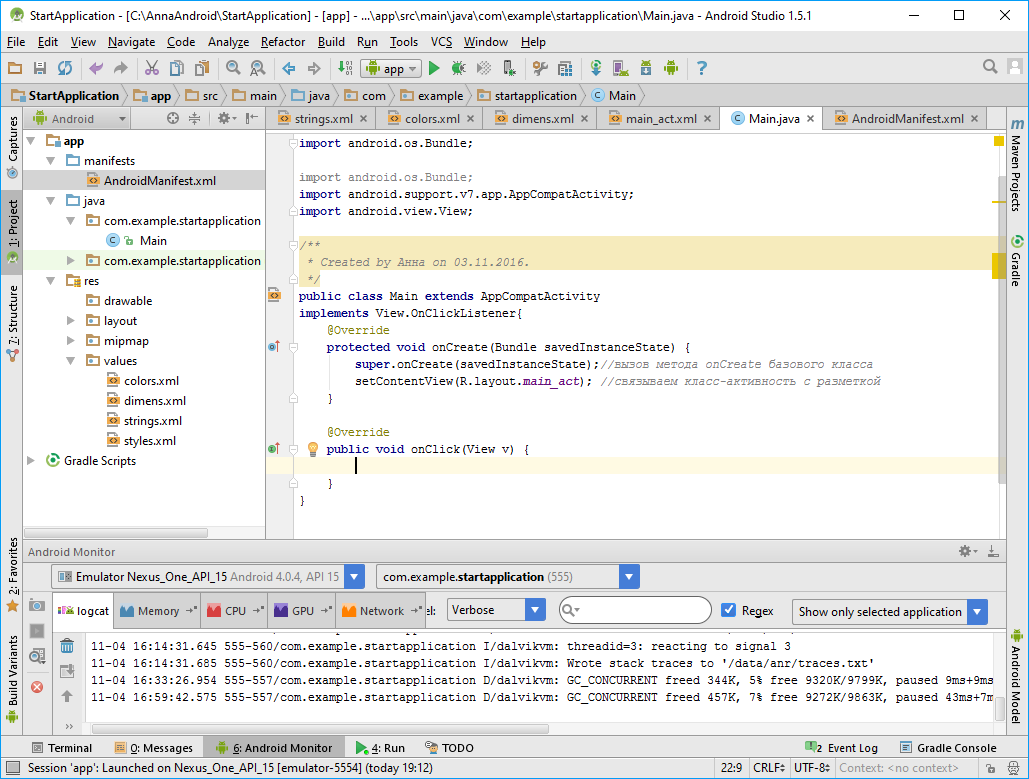
Нам сообщают, что класс ***Main*** либо должен быть объявлен абстрактным, либо в нем должен быть реализован метод ***onClick*** с параметром класса ***View*** – это метод интерфейса ***View.OnClickListener.*** Если мы щелкнем мышкой по красной сигнальной лампочке, то увидим всплывающую подсказку ***Click or press Alt-Enter***. После нажатия ***Alt-Enter*** появится выпадающий список. К котором мы должны выбрать опцию ***Implement methods*** (реализовать методы).



После выбора этой опции появится окно со списком методов интерфейса, которые требуется реализовать, для нашего интерфейса – это один единственный метод, который мы должны выбрать и нажать ***OK***.

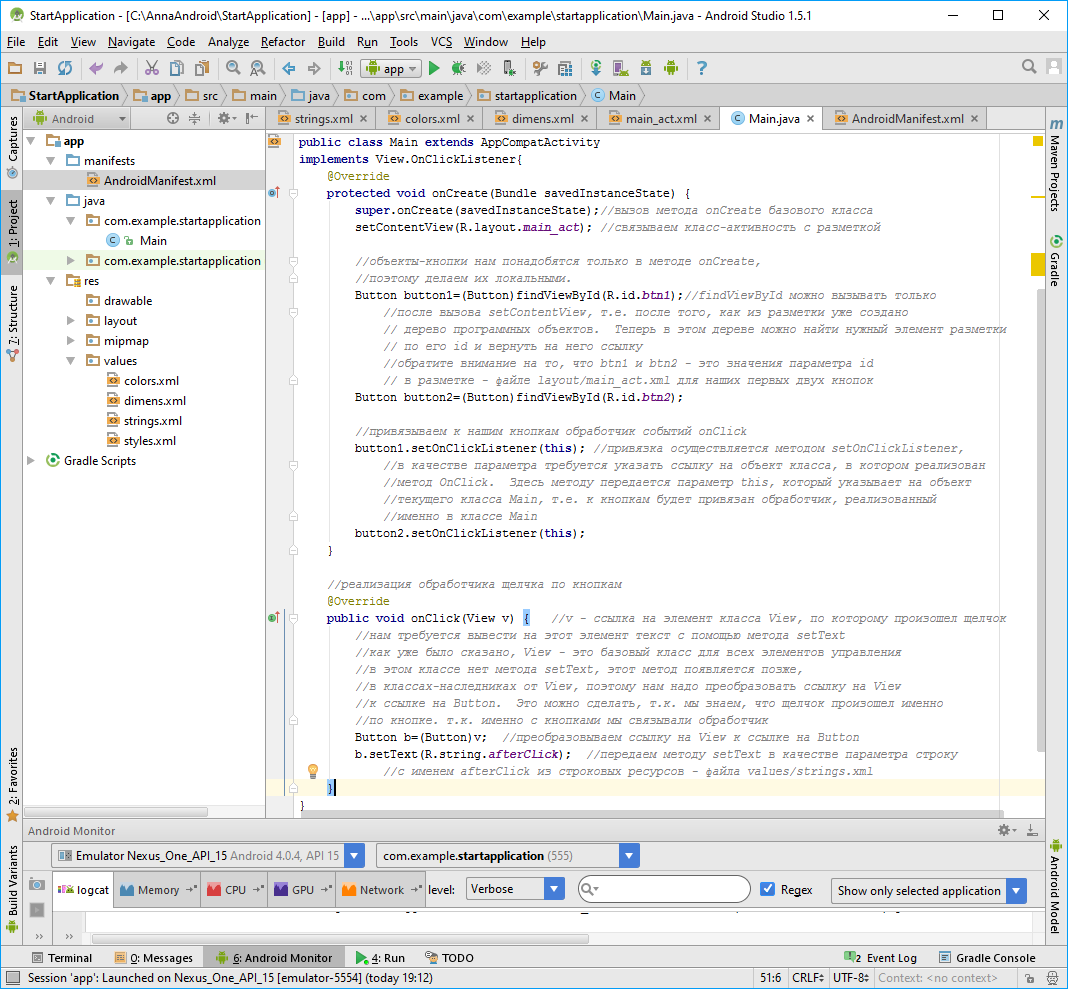


Android создаст заготовку для метода ***onClick***



Метод-обработчик ***onClick*** имеет параметр – ссылку на объект класса ***View,*** по которому произошел щелчок. ***View*** - это базовый класс для всех элементов управления, в том числе и для кнопок ***Button***. В нашем приложении нам понадобится только один обработчик ***onClick***, который мы привяжем ко всем кнопкам, с помощью параметра мы сможем узнать, по какой именно кнопке был щелчок и поменять надпись именно на этой кнопке. Если бы нам понадобилось более одного обработчика, мы уже не смогли бы использовать именно такой способ привязки.

Теперь нам надо создать программные объекты, соответствующие кнопкам, привязать к ним обработчик и написать код для обработчика.



Запускаем приложение (не закрывая эмулятор) и нажимаем на вторую кнопку, видим, что текст на ней поменялся:



Добавьте в приложение еще 2 кнопки, так, чтобы всего их оказалось 4.

**Задания для самостоятельного выполнения**

* Найдите в Проводнике папку, в которой Вы сохранили приложение, пройдите по пути

***app/src/main*** относительно корневой папки,

найдите там файлы с ресурсами, разметкой, кодом, манифестом, откройте их, например, в ***Notepad++,*** убедитесь, что там находится Ваш код

* В Android Studio поменяйте в ресурсах цвета, внешние и внутренние отступы, размер текста, Ваши строковые данные, посмотрите, как это будет выглядеть на макете и на эмуляторе при запущенном приложении
* Поменяйте в разметке ориентацию ***LinearLayout*** на ***horizontal***, посмотрите, как это будет выглядеть на макете и на эмуляторе при запущенном приложении
* Поменяйте в строковых ресурсах строку с названием приложения (на нее есть ссылка из манифеста), посмотрите, как это будет выглядеть
* Найдите файл ***res/values/styles.xml***, поменяйте там в стиле цвета (не меняя значения параметров ***name***), запустите приложение, посмотрите, что изменилось
* В Проводнике найдите в ресурсах файлы ***mipmap***, поменяйте немного изображения иконок в каком-либо графическом редакторе, убедитесь, что на эмуляторе поменялась иконка с вашим приложением

Преподаватель Яценко А.И.

(подпись)  
«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **2**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Привязка обработчика событий к элементу управления в разметке. Работа со строками. Поиск и исправление ошибок. Работа с цветом. Использование ресурса цвета. Изменение цвета текста и фона элементов управления в режиме запуска приложения.**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

# Проект 2. Обработчики событий в разметке

**Цель работы:**

* Привязка обработчика событий к элементу управления в разметке
* Работа со строками
* Работа с элементами управления
* Поиск и исправление ошибок

**Задание:**

Внешний вид активности

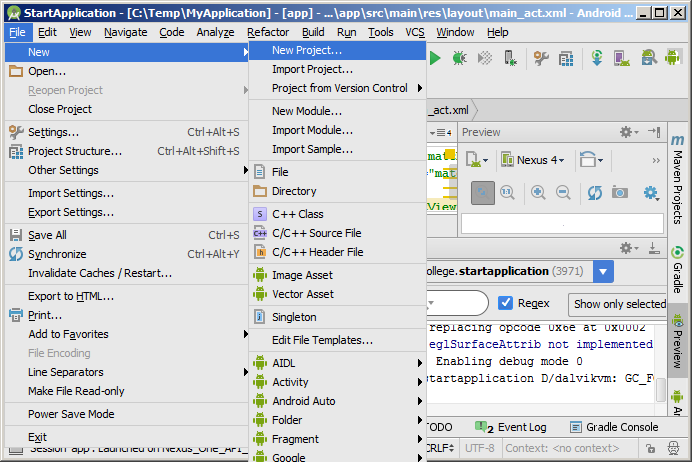


кнопки

По кнопке «Add \*» к содержимому ***TextView1*** добавляется 1 звездочка. По кнопке «Copy» содержимое одного ***TextView*** копируется в другой ***TextView***.

**Указания к выполнению задания:**

Начните новый проект. Это можно сделать и из уже открытой Android Studio:



Далее действуем так же, как и при создании нового проекта в прошлой работе.

Элемент разметки ***TextView*** соответствует элементу ***label*** в Delphi или Visual Studio – это надпись, содержимое которой пользователь не может изменять вручную в запущенном приложении.

Давайте сделаем так, чтобы наши элементы разметки занимали все окно по высоте, при этом размер ***TextView*** по вертикали был вдвое больше, чем размер кнопок ***Button***. Для этого мы в разметке используем нулевое значение (***0dp***) параметра ***height*** для всех элементов, это значение – признак того, что элемент занимает все свободное от других элементов пространство (в данном случае – по высоте). Если высота или ширина какого-либо элемента разметки нулевая, то для этого элемента становится обязательным параметр ***weight*** – вес – это простое числовое значение без наименований, указывающее взаимное соотношение высот для всех элементов с нулевым параметром.

Пример фрагмента разметки:

<**LinearLayout …  
 android:orientation="vertical"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"**>  
 <**TextView**

**...   
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:layout\_weight="2"**/>  
 <**TextView**

**...   
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:layout\_weight="2"**/>  
  
 <**Button  
 ...   
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:layout\_weight="1"**/>  
  
 <**Button  
 ...   
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:layout\_weight="1"**/>  
</**LinearLayout**>

Обратите внимание на то, что веса для ***TextView*** указаны в 2 раза большие, чем для ***Button***. Вместо троеточий укажите другие параметры тегов. Для того чтобы область ***TextView*** была видна, укажите для ***TextView*** цвет фона; чтобы элементы отделялись друг от друга, используйте внешние отступы. Так как нам придется обращаться к некоторым элементам разметки из кода, укажите для элементов параметр ***id*** (можно его указать для всех элементов).

Теперь перейдем к коду. Так как доступ к объектам – элементам разметки нам понадобится из разных методов класса, определим ссылки на объекты на уровне класса.

**public class** Main **extends** AppCompatActivity {  
 *//на уровне класса определяем ссылки на TextView и Button, пока эти ссылки пустые  
 //здесь мы не можем вызвать метод findViewById, который ищет и возвращает ссылку на объект* TextView **textView1**, **textView2**;  
 Button **buttonAdd**, **buttonCopy**;  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***main\_act***);  
 **textView1**=(TextView)findViewById(R.id.***tv1***); *//находим в дереве объектов объект TextView,  
 //возвращаем на него ссылку, преобразуем ссылку на View к ссылке на TextView  
 //и присваиваем эту ссылку textView1.  
 //Предполагается, что в файле разметки у одного из элементов TextView  
 //есть параметр id со значением tv1  
 //****!!!*** *Обратите внимание на возможную типичную ошибку, если мы напишем здесь оператор  
 //~~TextView~~ textView1=(TextView)findViewById(R.id.tv1);  
 //то создастся локальный объект textView1, который пропадет сразу после того,  
 //как отработает метод onCreate, а ссылка-поле textView1 на уровне класса  
 //останется пустой, соответственно, мы не сможем использовать, например,  
 //оператор textView1.setText(...) в каком-либо другом методе, кроме onCreate  
   
 //создайте таким же образом остальные объекты, соответствующие элементам разметки.* }  
}

В данной работе предполагается использование двух различных обработчиков для двух кнопок, поэтому мы не сможем их создать таким же образом, как в прошлой работе, давайте укажем имена обработчиков в разметке. Обратите внимание на то, что имена обработчиков в разметке можно указывать только для событий ***onClick***.

В разметке добавим для элементов ***Button*** параметры ***onClick*** с именами обработчиков:

<**Button**

**...   
 android:onClick="onAdd"**

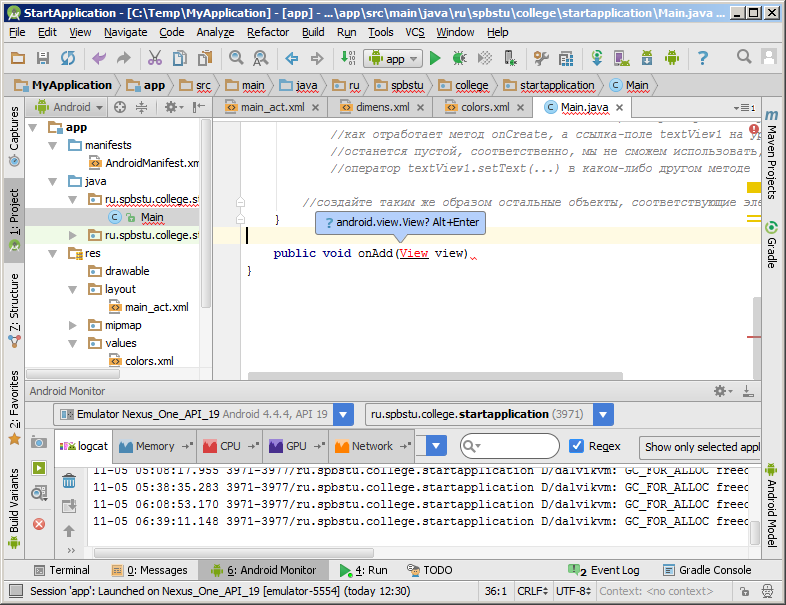
**...**/>  
  
<**Button**

**...  
 android:onClick="onCopy"** **...**

/>

Имена обработчиков мы можем придумывать сами.

Теперь создадим обработчики с такими именами и обязательным параметром ***View*** в классе ***Main***:



При написании заголовка обработчика может появиться указанное сообщение. Это происходит потому, что еще не подключена библиотека с соответствующим классом, нажмите ***Alt+Enter*** чтобы эту библиотеку подключить. В дальнейшем Вы будете часто видеть подобное сообщение.

Добавленные нами в класс заготовки для обработчиков:

**public void** onAdd(View view){  
   
}  
  
**public void** onCopy(View view){  
  
}

Теперь напишем тело обработчиков – операторы, которые будут выполняться, когда произойдет щелчок по соответствующей кнопке. Пусть первый обработчик копирует текст из ***textView1*** на первую кнопку, а второй обработчик копирует текст из ***textView2*** и первой кнопки на вторую кнопку, при этом добавляя еще и свой собственный текст. Это не совсем то, что требуется в задании на лабораторную работу, но это тот код, который поможет Вам выполнить Ваше задание.

**public void** onAdd(View view){  
 String s=**textView1**.getText().toString(); *//берем текст из textView1,   
 //преобразуем его в строку* **buttonAdd**.setText(s); *//копируем строку на первую кнопку*}

**public void** onCopy(View view){  
 String s1=**"вторая надпись: "** + **textView2**.getText().toString();  
 String s2=**", первая кнопка: "** + **buttonAdd**.getText().toString();  
 **buttonCopy**.setText(s1 + s2);  
}

Вид активности после нажатия на вторую кнопку:



Методы ***setText*** и ***getText*** для ***TextView*** работают так же, как и для ***Button***.

**Поиск и исправление ошибок**

1. **Пример ошибки времени выполнения**

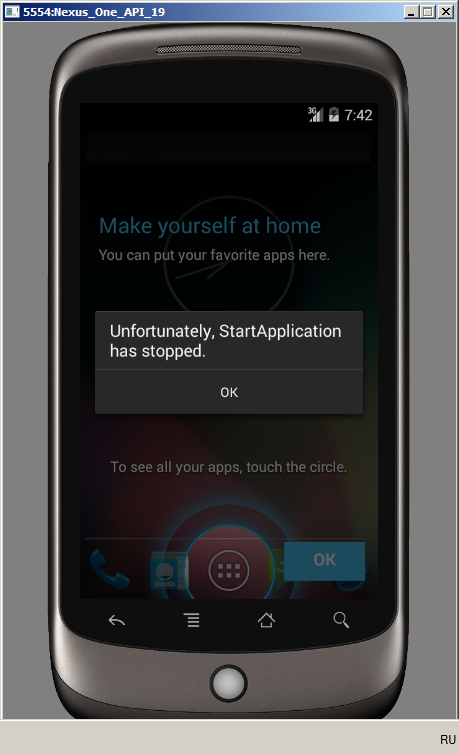
Допустим, что у нас в коде в методе onCreate вместо оператора

**textView1**=(TextView)findViewById(R.id.***tv1***);

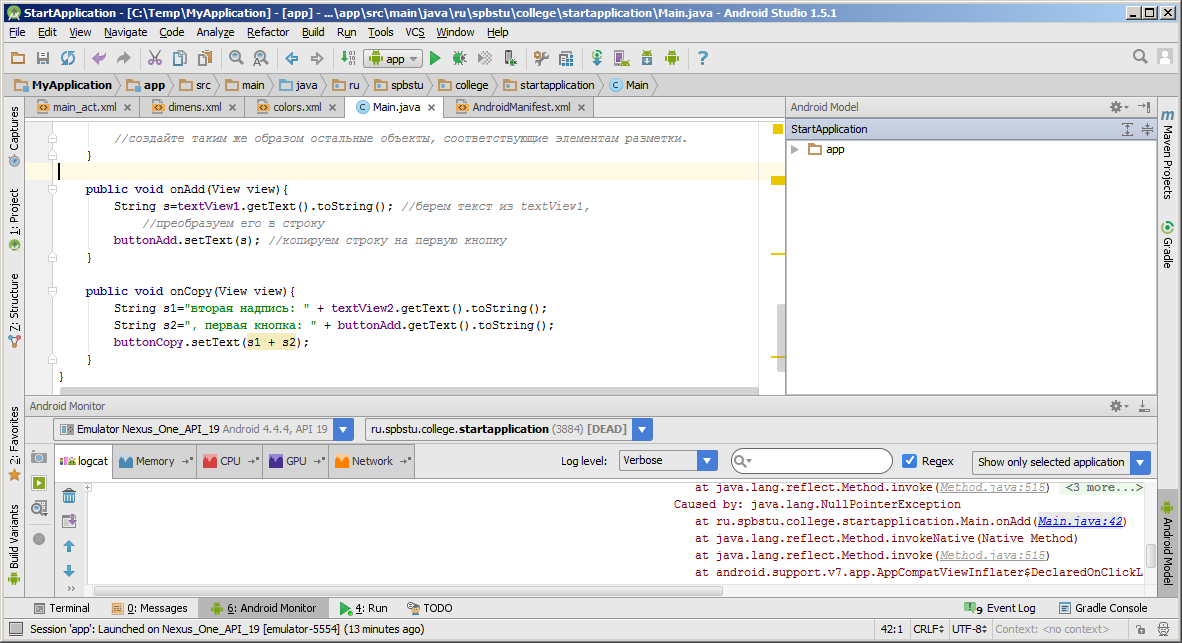
записан оператор

TextView textView1=(TextView)findViewById(R.id.***tv1***);

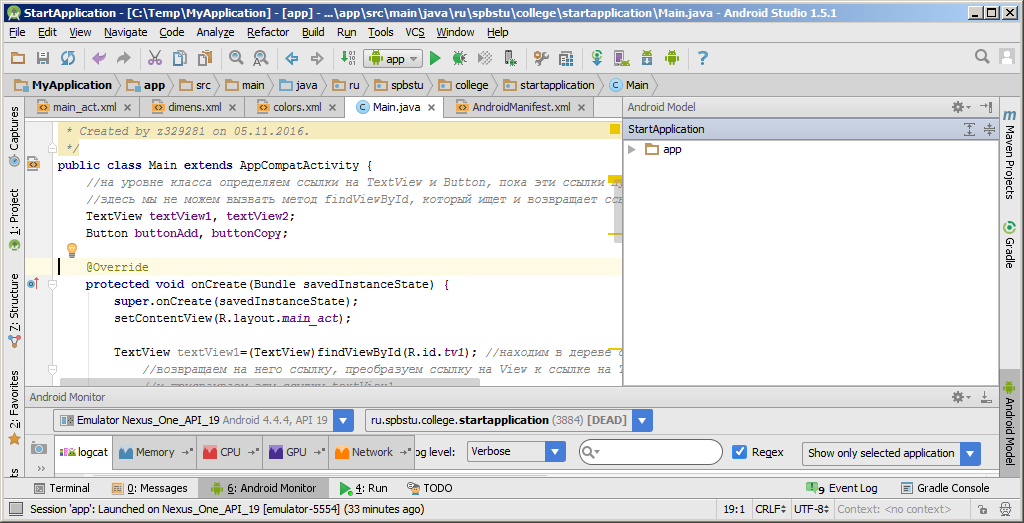
После запуска приложения при попытке нажать кнопку «***Add \****» мы увидим на нашем устройстве сообщение об ошибке:



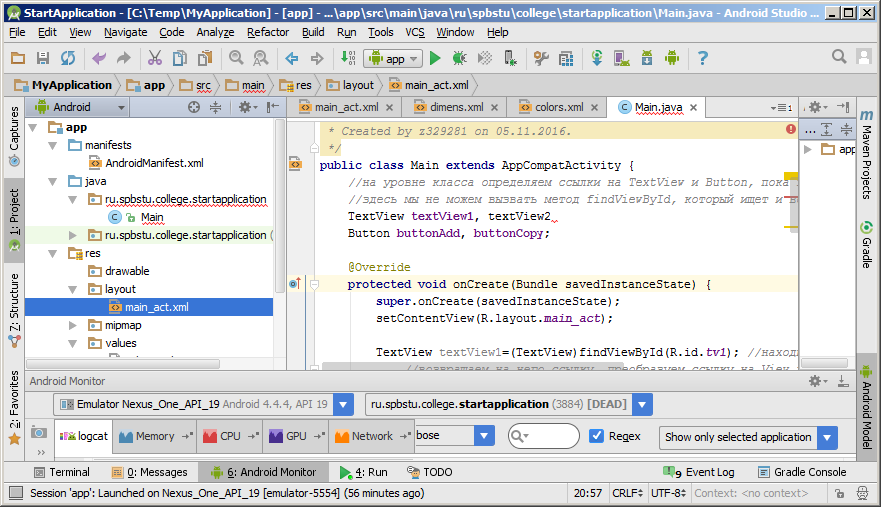
Определим, где произошла ошибка – на вкладке «***Android Monitor***» найдем ссылку синего цвета – она указывает на ошибку в нашем коде. Если мы щелкнем по этой ссылке, курсор укажет на место, где произошла ошибка. Можно также найти и прочитать сообщение об ошибке – в данном случае это – ***NullPointerException*** – исключение, вызванное попыткой обращения к объекту, на который отсутствует ссылка (ссылка пустая).



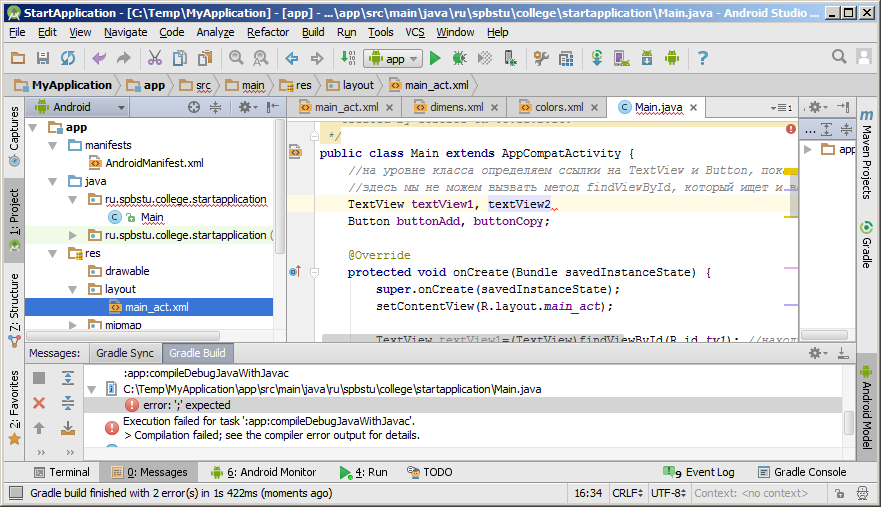
Обратите внимание на то, что в коде ссылка ***textView1*** обозначена серым цветом - признак того, что далее она нигде не используется (т.к. она локальная и действует только в пределах ***onCreate***).



1. **Пример синтаксической ошибки**

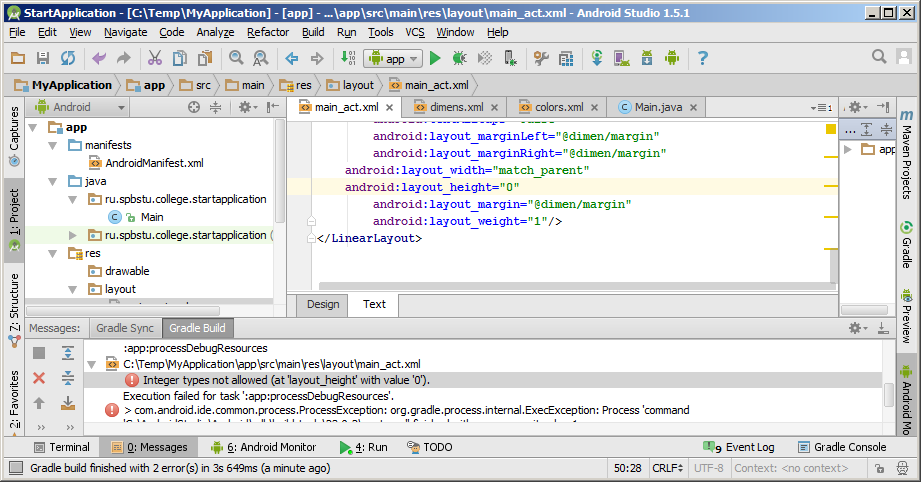


Допустим, у нас пропущена точка с запятой – в этом случае Android подчеркивает красным цветом предполагаемое место ошибки, а также вкладку с файлом, в котором обнаружена ошибка, и все папки вверх по иерархии, в которых содержится этот файл.

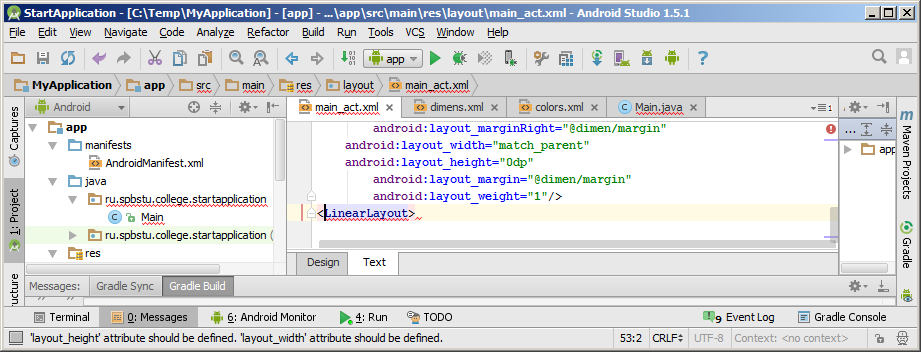


При попытке запустить приложение выдается сообщение об ошибке. При щелчке по сообщению курсор перемещается на ошибочный оператор.

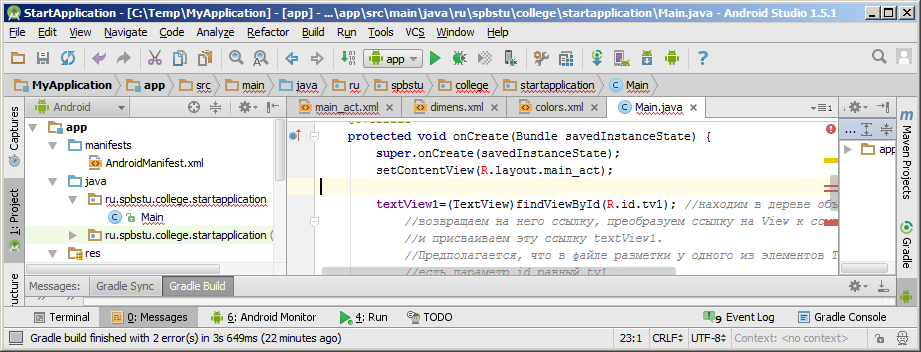
1. **Примеры ошибок разметки**



Здесь в качестве значения параметра высоты указано значение «***0***», а не ***«0dp»***. В сообщении об ошибке говорится о том, что целый тип здесь недопустим. Иногда ошибка проявляется только при попытке запуска приложения.



Здесь в закрывающем теге ***LinearLayout*** пропущен слеш.



Иногда при ошибках разметки имя класса-ресурса ***R*** выделяется красным цветом в коде. Если при этом нигде в разметке ошибка не выделена, надо запустить приложение, прочитать сообщение об ошибке и щелкнуть по этому сообщению, чтобы ошибку найти.

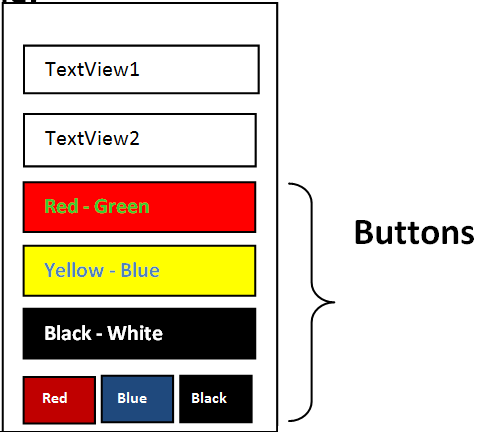
# Проект 3. Работа с цветом

**Цель работы:**

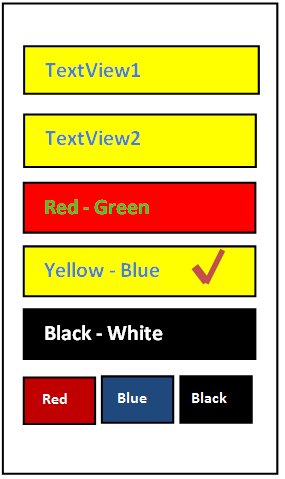
* Использование ресурса цвета
* Изменение цвета текста и фона элементов управления в режиме запуска приложения

**Задание:**

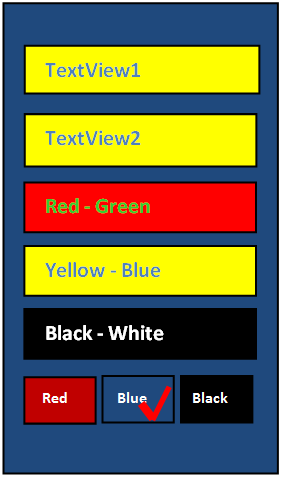
Внешний вид активности



При нажатии на любую из «длинных» кнопок цвет ее текста и фона должны установиться для обоих ***TextView*** (здесь мы видим, что при щелчке по кнопке с желтым цветом фона и голубоватым цветом текста соответствующие цвета установились для обоих ***TextView***):

****

При нажатии на любую из коротких кнопок цвет ее фона должен установиться для всей активности (здесь видно, что при щелчке по синей кнопке вся активность перекрасилась в синий цвет).

****

**Указания к выполнению задания:**

* Для формирования активности с подобным внешним видом можно использовать вложенные ***LinearLayout –*** во внешнем из них ориентация вертикальная, во внутреннем – горизонтальная (по умолчанию). Во внешнем ***LinearLayout*** с вертикальной ориентацией элементов расположены 6 элементов – 2 ***TextView***, 3 кнопки и вложенный ***LinearLayout.*** Во вложенном ***LinearLayout*** с горизонтальной ориентацией элементов (это ориентация по умолчанию) расположены 3 кнопки
* Для того, чтобы 3 горизонтально расположенные кнопки были одинаковой ширины, можно использовать значение параметра ширины (***layout\_width***) 0dp и значение параметра веса (***layout\_weight***) 1, подобно тому, как это было сделано во втором проекте для высоты.
* Для того чтобы применить перекраску фона для всей активности, потребуется в коде создать программный объект, соответствующий корневому элементу активности – внешнему ***LinearLayout.*** Для этого в разметке требуется указать идентификатор (***id***) для этого элемента.
* Для изменения цвета фона обоих ***TextView*** потребуется в коде создать соответствующие им программные объекты, для этого в разметке надо будет указать идентификаторы (***id***) для этих элементов.
* В разметке указываем имена обработчиков события ***onClick*** для кнопок

Фрагмент разметки:

<**LinearLayout ...**

**android:id="@+id/mLL"   
 android:orientation="vertical"**

**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"**>  
 <**TextView ...** />

<**TextView ...** />  
 <**Button ...** />

<**Button ...** />

<**Button ...** />

<**LinearLayout  
 android:orientation="horizontal"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"**>  
 <**Button**

**...  
 android:layout\_width="0dp"  
 android:layout\_weight="1"  
 android:layout\_height="wrap\_content"** />  
 <**Button**

**...  
 android:layout\_width="0dp"  
 android:layout\_weight="1"  
 android:layout\_height="wrap\_content"** />

<**Button**

**...  
 android:layout\_width="0dp"  
 android:layout\_weight="1"  
 android:layout\_height="wrap\_content"** />  
 </**LinearLayout**>  
</**LinearLayout**>

* Так как соответствующие элементам разметки объекты нам понадобятся в различных обработчиках, определим ссылки на них на уровне класса
* В методе ***onCreate*** создаем объекты, адреса объектов присваиваем ссылкам
* В обработчиках для кнопок перекрашиваем элементы

**public class Main extends AppCompatActivity{**

**private LinearLayout linearLayout;**

**private TextView textView1, textView2;**

…

**public void onCreate(Bundle savedInstanceState) {**

**super.onCreate(savedInstanceState);**

**setContentView(R.layout.main\_activity);**

**linearLayout=(LinearLayout)findViewById(R.id.mLL);***//создаем объект LinearLayout, соответствующий корневому элементу разметки*

**textView1=...**

**textView2=...**

**...**

**}**

**public void toRed(View view){**

**linearLayout.setBackgroundColor(ContextCompat.getColor(this, R.color.myRedColor));** *//красим активность (ее корневой элемент) в цвет из ресурса цветов*

**}**

**...**

**}**

Для изменения цвета фона элемента используем метод ***setBackgroundColor,*** для изменения цвета текста используем метод ***setTextColor*.**

Можно извлекать цвет из ресурса цветов различными способами:

* **getResources().getColor(R.color.myRedColor)**
* **getResources().getColor(R.color.myRedColor,null)**
* **ContextCompat.getColor(this, R.color.myRedColor)**

Можно использовать стандартные цвета, например, красим фон элемента ***myEl*** в один из стандартных цветов:

**myEl.setBackgroundColor(Color.CYAN);**

Можно использовать цвета в формате ARGB, например, красим текст элемента ***myEl*** в цвет, задаваемый этим форматом:

**myEl.setTextColor(0xff00ffaa);**

Преподаватель Яценко А.И.

(подпись)  
«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **3**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Выполнение простого расчетного задания. Способы отладки приложения.**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

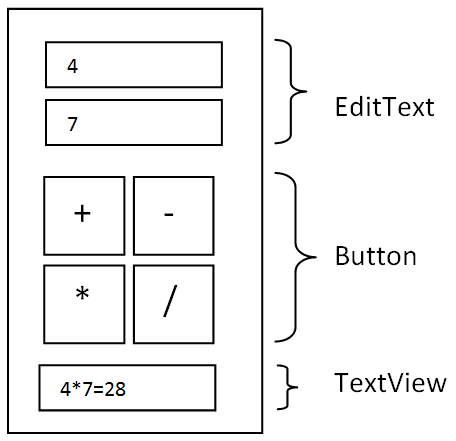
# Проект 4. Простейший калькулятор и отладка приложения

**Цель работы:**

* Выполнение простого расчетного задания
* Преобразование типов данных
* Знакомство со способами отладки приложения

**Задание:**

Вид активности:



Пользователь вводит целые числа в ***EditText***, затем нажимает на кнопку со знаком действия, в ***TextView*** при этом отображается соответствующий пример с ответом.

**Указания к выполнению задания:**

***EditText*** – элемент управления, содержимое которого пользователь может изменять в режиме запуска приложения. Аналоги – ***TextBox*** в Visual Studio или ***Edit*** в Delphi. В разметке для размещения этого элемента следует использовать одноименный тег.

Рассмотрим следующий фрагмент кода, в котором демонстрируется использование ***EditText*** для выполнения простого расчета:

**//et – объект класса EditText**

**//tv – объект класса TextView**

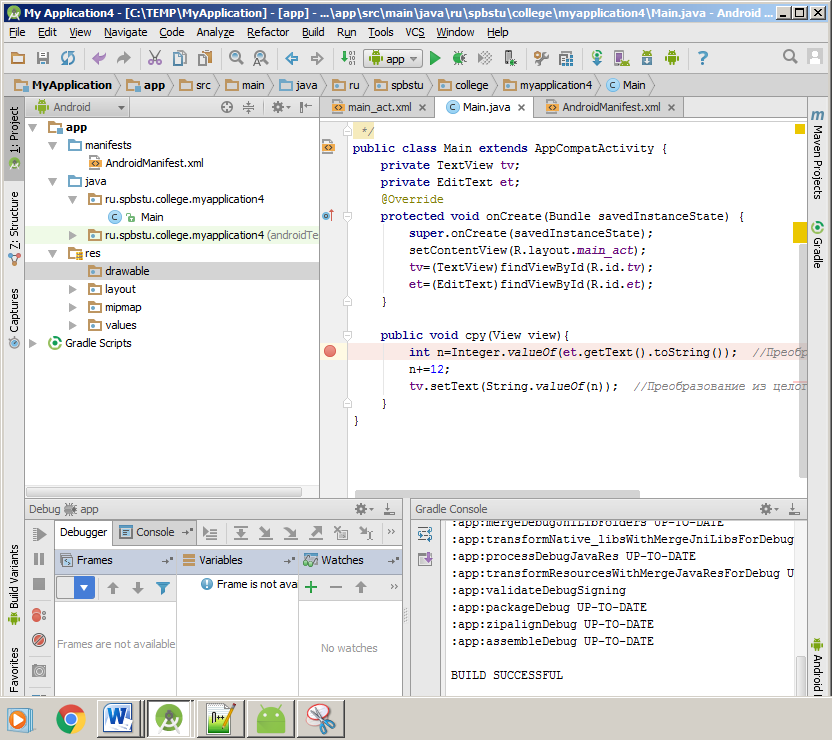
**int** n=Integer.*valueOf*(**et**.getText().toString()); //*Преобразование из строкового типа в целый*  
n+=12;  
tv.setText(String.*valueOf*(n)); //*Преобразование из целого типа в строковый*

**Работа с отладчиком**

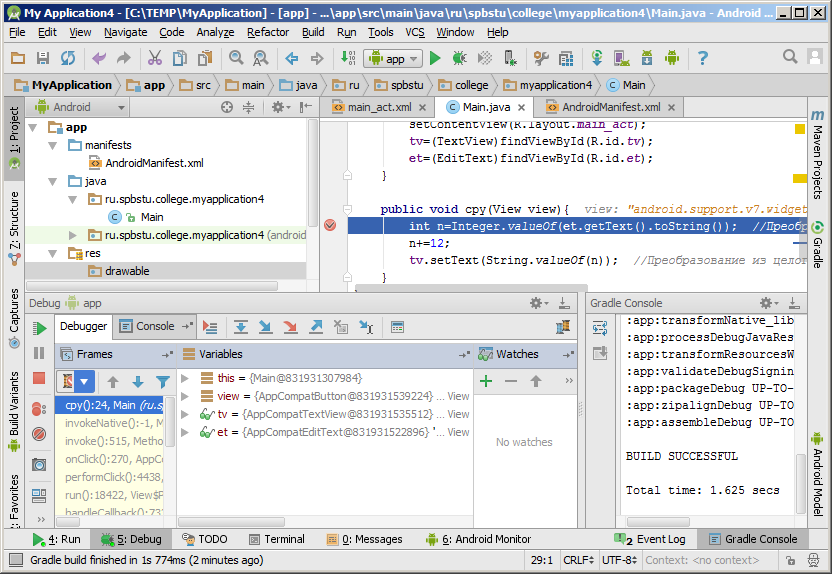
Также как и в других средах программирования, в Android Studio есть возможность отлаживать приложение с помощью отладчика. Напоминание: с помощью отладчика можно

* остановить программу в любой точке
* выполнять программу в пошаговом режиме
* просмотреть значение любой переменной или подсчитать значение выражения в любой момент времени

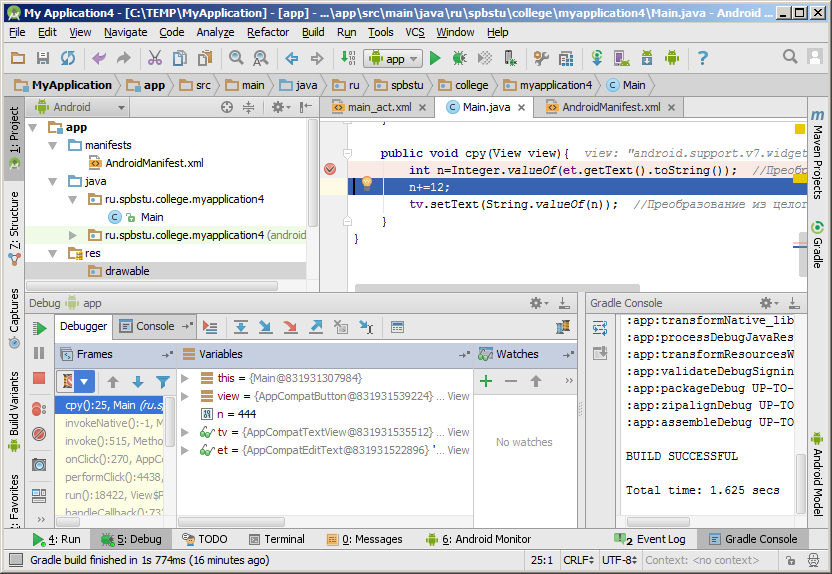
Укажите точку останова щелчком мыши слева от оператора, на котором Вы хотите остановить программу, затем нажмите значок отладки:



Если точка останова установлена внутри обработчика нажатия на кнопку (как в нашем случае), после запуска приложения нажмем на эту кнопку, чтобы программа добралась до точки останова. Далее мы увидим текущие значения переменных и инструменты для работы в отладчике. Подведите курсор к соответствующим значкам и посмотрите, для чего каждый из них предназначен.



Выполним 2 шага в пошаговом режиме и проследим за значением локальной переменной ***n***. Здесь указано значение ***n***, извлеченное из ***EditText***, в дальнейшем это значение увеличится на 12.



**Отладка с помощью промежуточной печати**

Мы можем использовать для отладки промежуточную печать – свои собственные отладочные сообщения, которые выводятся на вкладку ***logcat*** вместе с системными сообщениями. С помощью отладочных сообщений можно, например, организовать трассировку - проследить, куда наша программа добралась, а куда – нет.

**public class** Main **extends** AppCompatActivity {  
 **private** TextView **tv**;  
 **private** EditText **et**;  
  
 **private static final** String ***TAG*** = **"Main"**;  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***main\_act***);  
 **tv**=(TextView)findViewById(R.id.***tv***);  
 **et**=(EditText)findViewById(R.id.***et***);  
 }  
  
 **public void** cpy(View view){  
 Log.*d*(***TAG***, **"in cpy"**); *//Для вывода отладочных сообщений используем класс Log*

*//d – используется именно для наших отладочных сообщений,*

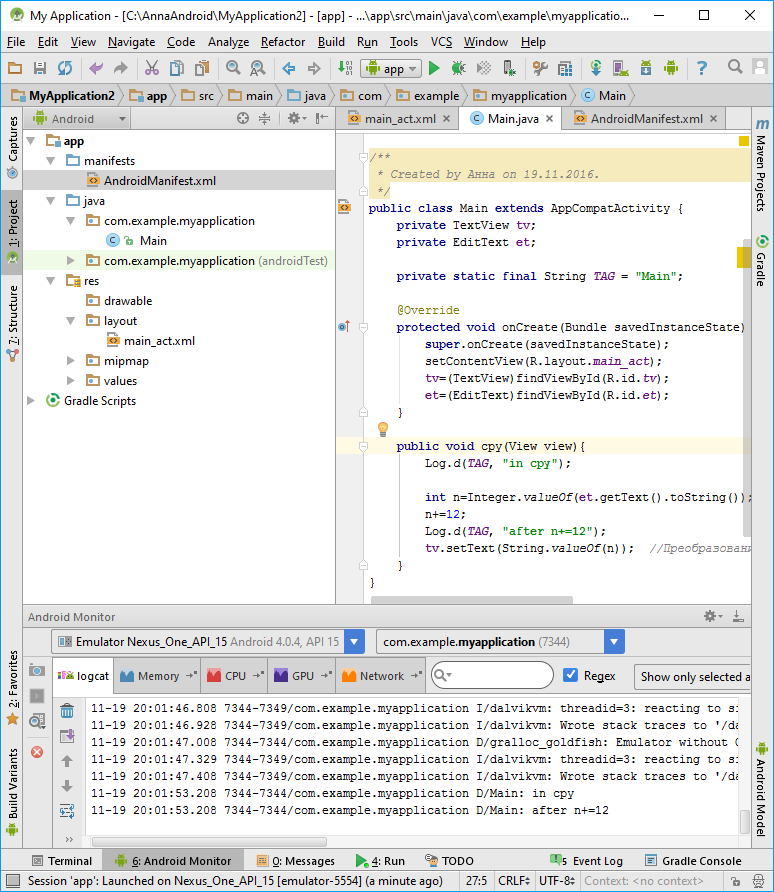
*//характеризует уровень приоритета*

*//первый параметр – строка, в которой рекомендуется указывать*

*//класс – источник сообщения, здесь – TAG=”Main”*

*//второй параметр – строка – текст сообщения*  
  
 **int** n=Integer.*valueOf*(**et**.getText().toString()); *//Преобразование из строкового типа в целый* n+=12;  
 Log.*d*(***TAG***, **"after n+=12"**);  
 **tv**.setText(String.*valueOf*(n)); *//Преобразование из целого типа в строковый* }  
}

Здесь мы видим наши отладочные сообщения “***D/Main: in cpy***” и “***D/Main: after n+12***”, значит, по крайней мере до этих операторов наша программа добралась.



Преподаватель Яценко А.И.

(подпись)  
«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **4**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Разработка приложения с использованием элемента ListView и стандартного строкового адаптера**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

# Проект 5. Работа со списком

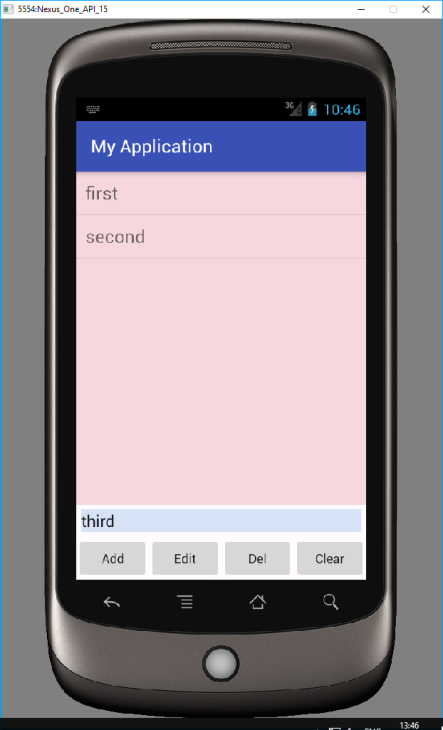
**Цель работы:**

* Знакомство с элементом ***<ListView>***
* Знакомство со стандартным адаптером
* Блокировка кнопок

**Задание:**

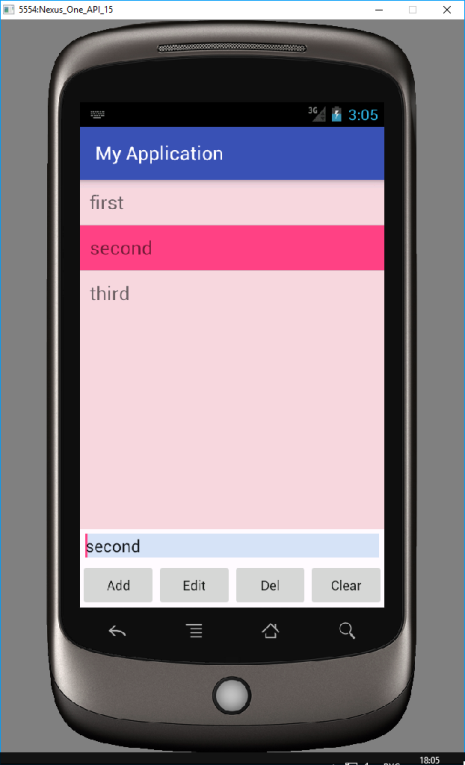
Вид активности:

* розоватым цветом выделен элемент ***ListView***, предназначенный для вывода элементов списка. На рисунке в списке уже есть 2 элемента – «***first***» и «***second***».
* синеватым цветом выделен элемент ***EditText***
* в нижнем ряду – 4 кнопки



Функциональность:

* при щелчке по ***Add*** строка из ***EditText*** должна добавиться в ***ListView***
* при щелчке по пункту списка этот пункт должен подкрашиваться другим цветом (меняем цвет фона), если при этом ранее какой-то другой пункт списка был выбран и выделен цветом, для этого пункта восстанавливается первоначальный цвет
* при щелчке по пункту списка соответствующая строка должна скопироваться в ***EditText***, где ее можно будет впоследствии редактировать
* при щелчке по ***Edit*** находящаяся в ***EditText*** отредактированная строка замещает текущий элемент списка, если такой имеется (текущий элемент списка – это выделенный цветом элемент, по которому ранее щелкали).
* при щелчке по ***Del*** текущий элемент списка (если он имеется) удаляется
* при щелчке по ***Clear*** список очищается



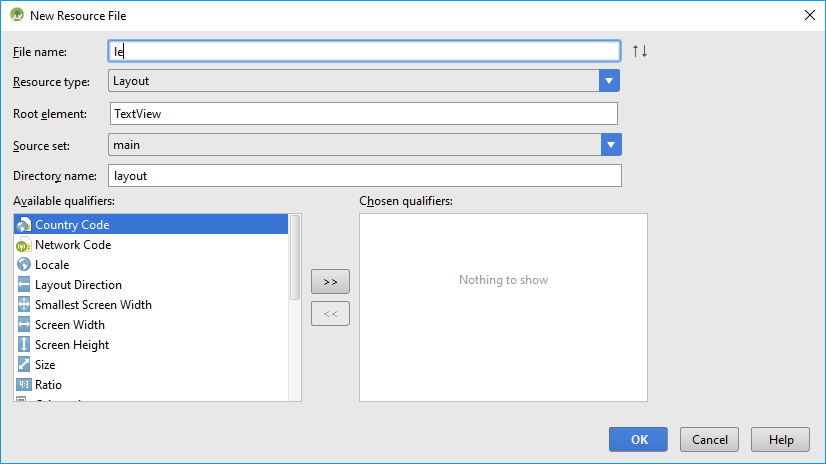
**Указания к выполнению задания:**

В разметке главной активности для ***ListView*** можно указать высоту ***0dp***, чтобы по высоте список занимал все свободное (оставшееся от других элементов) пространство.

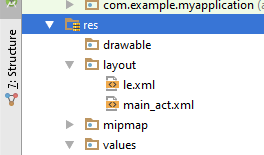
<**ListView  
 android:id="@+id/lw"  
 android:background="@color/lw"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:layout\_weight="1"** />

Для ***ListView,*** ***EditText,*** кнопок ***Add*** и ***Del*** укажите параметр ***id***, так как к этим элементам потребуется обращаться из кода.

Нам потребуется создать отдельную разметку для элемента списка. Так как наш список будет содержать строки, то для элемента списка подойдет ***TextView*** – это будет корневой и единственный элемент нашей отдельной разметки. Файл с разметкой создаем точно так же как и файл с разметкой для главной активности, только в поле «***корневой элемент***» пишем ***TextView***.



Теперь в папке layout будут 2 файла с разметкой



Разметка для элемента списка будет выглядеть примерно как на фрагменте снизу. Для элемента ***TextView*** можно указать отступы, цвета, шрифт.

<**TextView xmlns:android="http://schemas.android.com/apk/res/android"  
 android:padding="@dimen/padding"  
 android:textSize="@dimen/tS"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"**>  
</**TextView**>

Работа со списком обычно происходит через ***адаптер*** – объект специального класса, предназначенного для контроля за корректным отображением элементов списка. Для каждого списка можно установить свой адаптер. С помощью методов адаптера элементы добавляются в конец или внутрь списка, удаляются из списка и т.д.

Можно воспользоваться стандартным адаптером или написать свой собственный адаптер. Так как в данной работе мы собираемся помещать внутрь списка простые строки, можно обойтись уже готовым стандартным строковым адаптером. Адаптерам для работы требуется ресурс с разметкой для элемента списка. Для стандартного строкового адаптера корневым и единственным элементом такой разметки должен быть ***TextView***. Именно такую разметку мы и создали для нашего будущего адаптера чуть ранее.

Будем выполнять работу постепенно – вначале добьемся того, чтобы при запуске приложения в списке уже отображалась пара элементов - строк. Для этого в классе, связанном с главной активностью, поместим следующий код:

**public class** Main **extends** AppCompatActivity {  
 **private** ArrayAdapter<String> **arad**; *//ссылка на стандартный строковый адаптер –*

*//объект класса* ***ArrayAdapter<String>*** **private** ListView **lw**; *//ссылка на ListView  
 … //ссылка на EditText*  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 *… //связываем класс-активность с разметкой*

*… //получаем ссылку на* ***EditText***

**arad**=**new** ArrayAdapter<String>(**this**,R.layout.***le***); *//создаем стандартный строковый адаптер,*

*//второй параметр конструктора – файл*

*// с разметкой для элемента списка*

**lw**=(ListView)findViewById(R.id.***lw***); *//получаем ссылку на* ***ListView*** **lw**.setAdapter(**arad**); *//устанавливаем для списка адаптер*  
 **arad**.add(**"first"**); *// через метод* ***add*** *адаптера добавляем в список первую строку*

**arad**.add(**"second"**);   
 }  
}

Теперь запустите приложение и убедитесь, что в списке находятся 2 первые строки. Кнопки пока еще не должны работать.

Следующий шаг: напишите обработчик для кнопки ***Add*** – добавление новой строки в конец списка. Строка должна извлекаться из элемента ***EditText*** (***et.getText().toString()***, где ***et*** – имя объекта ***EditText***. Обратите внимание на то, что метод ***toString*** возвращает строку, которую еще требуется где-то сохранить) и добавляться в список с помощью метода ***add*** связанного со списком адаптера. Запустите приложение, убедитесь, что кнопка ***Add*** работает.

Далее напишите обработчик для кнопки ***Clear*** – очистка всего списка (***arad.clear()***). Запустите приложение, убедитесь, что кнопка ***Clear*** работает.

Теперь добьемся того, чтобы при щелке по элементу списка соответствующая строка скопировалась бы в ***EditText*** для возможного последующего редактирования. Для этого нам надо будет написать обработчик для щелчка по элементу списка. С этой целью реализуем в классе ***Main*** интерфейс ***AdapterView.*OnItemClickListener**. В проекте 1 мы уже имели дело с интерфейсами.

**public class** Main **extends** AppCompatActivity  
 **implements** AdapterView.OnItemClickListener{

…

}

Нам будет предложено реализовать следующий метод указанного интерфейса:

**public void** onItemClick(AdapterView<?> parent, View view, **int** position, **long** id) {}

Это и есть обработчик для элемента списка. Параметры обработчика:

* ***parent*** – ссылка на родителя, которым является ***ListView*** (нам пока не понадобится)
* ***view*** – ссылка на объект, соответствующий элементу разметки, по которому произошел щелчок
* ***position*** – целочисленный номер элемента в списке
* ***id*** – целочисленный идентификатор элемента в списке (нам пока не понадобится)

Напишем тело обработчика:

String s=**arad**.getItem(position); *//метод* ***getItem*** *адаптера, связанного со списком,*

*//по элементу которого щелкнули, возвращает содержимое*

*//элемента списка, в данном случае – строку, по ее номеру*

… *//поместим строку в* ***EditText***

В методе ***onCreate*** установите для списка обработчик щелчка по его элементу: ***lw.setOnItemClickListener(this);*** В качестве параметра метода ***setOnItemClickListener*** требуется указать ссылку на объект класса, в котором находится обработчик. Обработчик находится в классе Main (это пока наш единственный класс) – в том же самом классе, в котором вызывается указанный метод, следовательно, в качестве параметра можно указать ***this*** – ссылку на объект текущего класса ***Main***. Запустите приложение, щелкните по элементу списка, убедитесь, что код работает.

Если мы отредактировали строку и хотим теперь ее заменить по нажатию кнопки Edit, окажется, что неизвестно, какую по номеру строку нужно менять, так как позиция строки была параметром обработчика щелчка по элементу списка и, как только выполнение этого обработчика завершилось, позиция оказалась утерянной. Следовательно, целочисленную позицию последней выбранной строки полезно сохранить на уровне класса, чтобы потом ее использовать при редактировании (в обработчике кнопки ***Edit***). Можно также хранить на уровне класса ссылку на саму последнюю выбранную строку. Сделайте это самостоятельно.

Напишем теперь обработчик для ***Edit***.

String s = **arad**.getItem(**curP***);//получаем ссылку на последнюю выбранную из списка строку (все строки*

*//из списка хранятся в адаптере).*

*//****curP*** *– сохраненная на уровне класса позиция последней выбранной строки***arad**.remove(s); *//удаляем строку, на которую указывает ссылка, используя метод адаптера* ***remove*****arad**.insert(**et**.getText().toString(), **curP**); *//вставка новой версии строки из* ***EditText***

*//на место только что удаленной.*

*//Используется метод адаптера* ***insert***

Запустите приложение и проверьте, что редактирование работает.

Теперь напишите самостоятельно обработчик для кнопки ***Del*** – удаление последней выбранной строки. Проверьте, что удаление работает.

А что будет, если ни одна строка не была выбрана, а мы нажали на кнопки ***Edit*** или ***Del***? В этом случае могут возникнуть проблемы. Поэтому давайте заблокируем эти 2 кнопки до тех пор, пока строка не будет выбрана: ***editButton.setEnabled(false);*** Здесь ***editButton*** – это имя объекта-кнопки, которую мы блокируем. Конечно же, объекты – кнопки надо предварительно создать. Подумайте, где нужно заблокировать и разблокировать кнопки.

Попробуем подкрасим элемент списка, по которому щелкнули. Способ, которым мы это сделаем, не очень хороший, но воспользуемся пока таким. В обработчике щелчка по элементу списка запишите следующий оператор: ***view.setBackgroundColor(…)***. В скобках укажите выбранный цвет подкраски. ***view*** в этом операторе – это параметр обработчика – ссылка на элемент разметки для выбранной строки. Убедитесь, что подкраска работает.

Далее, как только мы выбрали и покрасили новый элемент, надо перекрасить обратно предыдущий выбранный элемент, и тут опять возникает вопрос, откуда мы возьмем на ссылку на предыдущий элемент и что делать, если предыдущего выбранного элемента пока еще не было. Можно поступить следующим образом: сохранять ссылку на выбранный элемент в переменной, определенной на уровне класса:

private View curView=null; *//при инициализации указываем пустую ссылку, далее в обработчике*

*//щелчка по элементу списка проверяем, если* ***curView==null****,*

*//значит, пока не существует предыдущего выбранного элемента,*

*//и, следовательно, не требуется восстанавливать его цвет,*

*//а если ссылка* ***curView*** *не пустая, значит, она содержит адрес*

*//предыдущего выбранного элемента, для которого надо*

*//восстановить цвет.*

*//Подумайте, где еще надо записать оператор* ***curView=null;***

Запустите готовое приложение, проверьте его работоспособность.

Преподаватель Яценко А.И.

(подпись)  
«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **5**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Разработка приложения с использованием нескольких активностей. Использование объекта Intent. Разработка приложения с несколькими активностями и списками.»

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

# Проект 6. Работа с несколькими активностями

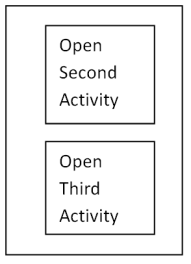
**Цель работы:**

* Создание нескольких активностей
* Запуск одной активности из другой

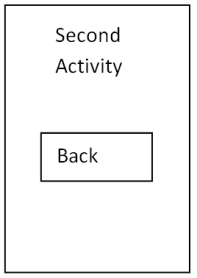
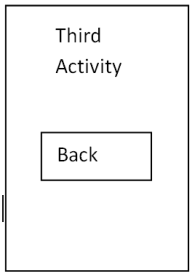
**Задание:**

На главной активности находятся кнопки «***Open Second Activity***» и “***Open Third Activity***”, по которым открываются вторая и третья активности соответственно. Вторая и третья активности содержат текст (***TextView***) «***Second Activity***» и «***Third Activity***» и кнопки «***Back***», по которым происходит возврат к главной активности.

Вид главной активности:

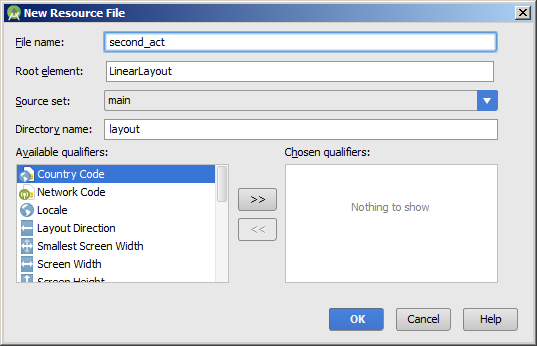


Виды второй и третьей активностей:

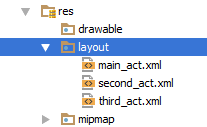
 

**Указания к выполнению задания:**

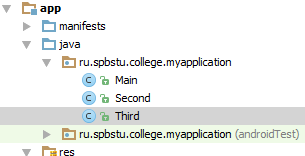
Для трех активностей создайте три файла разметки. Имена файлов пишем маленькими латинскими буквами без пробелов. Для всех трех активностей укажите в качестве корневого элемента ***LinearLayout.*** Расположите в разметках элементы так, чтобы активности приобрели указанный вид.



Теперь папка ***layout*** должна выглядеть так:



Для всех трех активностей создайте соответствующие классы (названия классов в ***java*** принято начинать с большой буквы). Теперь папка ***java*** будет выглядеть примерно так:



Каждый связанный с активностью класс должен быть наследником от базового класса-активности, например, ***AppCompatActivity,*** для того, чтобы унаследовать функциональность активности. В каждом классе-активности требуется перегрузить метод ***onCreate*** с одним параметром. В обработчике ***onCreate*** надо вызвать одноименный метод базового класса и вызвать метод ***setContentView*** для связи активности с разметкой (все это мы уже делали для главной активности).

Таким образом, класс для второй активности будет выглядеть примерно так:

public class Second extends Activity {

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.second\_act); *//связываем класс для второй активности с разметкой*

*//для второй активности, находящейся в файле* ***second\_act.xml***

}

}

Теперь рассмотрим, как из одной активности вызвать другую. Это делается через объект класса ***Intent*** (намерение). Рассмотрим обработчик для кнопки ***Open Second Activity***:

public void openSecondActivity(View view){

Intent intent = new Intent(this, Second.class); *//создаем объект класса* ***Intent****, в качестве*

*//второго параметра передаем ему указание*

*//класс активности, которую требуется открыть*

startActivity(intent); *//запускаем активность. Методу* ***startActivity*** *передаем параметр – объект*

*//класса* ***Intent****, в котором содержатся сведения о классе запускаемой*

*//активности*

}

Для возврата со второй активности обратно на главную по кнопке ***Back*** не надо использовать объект ***Intent***, можно просто вызвать метод ***onBackPressed();***

Теперь откройте манифест. Манифест представляет наше приложение операционной системе и должен содержать сведения обо всех его активностях. Сведения о главной активности добавим так, как мы это делали всегда, кроме этого, добавим внутрь тега ***application*** сведения о второй активности:

<**activity android:name=".Second"**/>

С третьей активностью работаем аналогично.

# Проект 7. Работа с несколькими активностями и списком одновременно

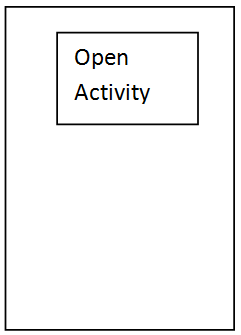
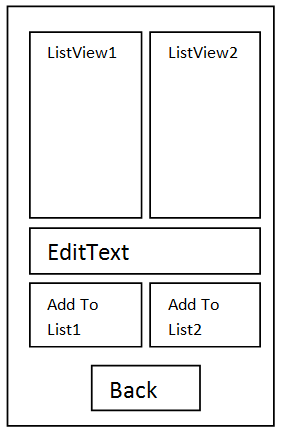
**Цель работы:**

* Использование в одном приложении нескольких активностей и нескольких списков

**Задание:**

Главная активность содержит кнопку «***Start Activity***», по которой открывается вторая активность. Вторая активность содержит 2 ***ListView***, ***EditText***, 2 кнопки «***Add To List 1***» и «***Add To List 2***» и кнопку «***Back***». По кнопкам «***Add To List 1***» и «***Add To List 2***» содержимое ***EditText*** добавляется в списки ***ListView 1*** и ***ListView 2*** соответственно. По кнопке «***Back***» происходит возврат в главную активность.

Вид главной и второй (вызываемой) активностей:

**Указания к выполнению задания:**

Мы уже прошли все, что требуется для выполнения данной работы, надо только обратить внимание на то, что для каждого списка требуется свой адаптер, т.е. всего должно быть 2 адаптера. Можно использовать одну и ту же разметку для элемента списка для обоих списков.

Преподаватель Яценко А.И.

(подпись)  
«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **6**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Разработка приложения с использованием ресурсов-массивов. Разработка приложения с использованием настроек (предпочтений пользователя) для хранения информации. Представление о жизненном цикле активности»

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

# Проект 8. Ресурсы-массивы

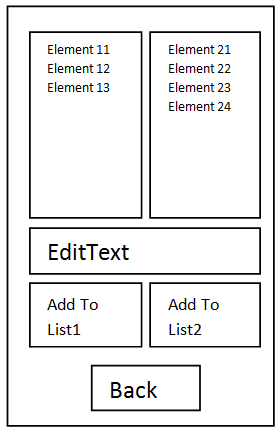
**Цель работы:**

* Использование ресурсов-массивов строк

**Задание:**

Доработать задание проекта 7 так, чтобы при загрузке второй активности данные для обоих списков брались из строковых массивов, элементы которых хранятся в ресурсах. При открытии второй активности оба списка уже должны быть заполнены строками из массивов-ресурсов, содержимое списков должно быть различное. Далее заполняем списки с помощью кнопки Add также как в предыдущей работе.

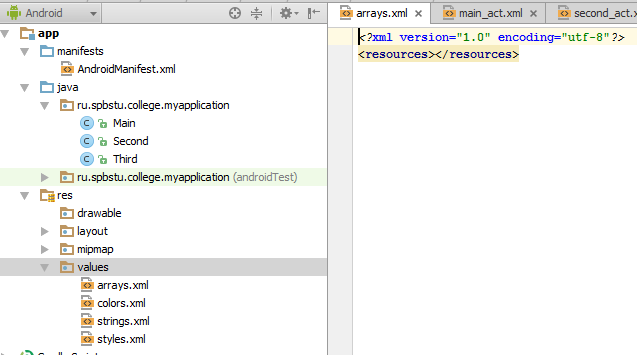
Примерный вид второй активности сразу же после ее открытия:



**Указания к выполнению задания:**

Массивы-строки поместите внутрь тега ***resources***  одного из ***xml***-файлов в папке ***values***. Можно использовать, например, файл ***strings.xml*** или создать отдельный файл, например, ***arrays.xml***





**<resources>  
 <string-array name="ar1">  
 <item>Yes1</item>  
 <item>Yes2</item>  
 <item>Yes3</item>  
 <item>Yes4</item>  
 </string-array>**

**<string-array name="ar2">  
 ...**

**</string-array>  
</resources>**

Восстановление массива из ресурса в коде и добавление строк из массива в список через адаптер:

**String []ar1=getResources().getStringArray(R.array.*ar1*);**

**myArAd1.addAll(ar1);**

# Проект 9. Настройки

**Цель работы:**

* Использование настроек (предпочтений пользователя) для сохранения и восстановления небольшого объема информации.
* Представление о жизненном цикле Android-приложения

**Задание:**

На главной активности находятся 3 элемента ***EditText***, предназначенных для ввода информации пользователем. Считаем, что это информация для 2 строковых элементов настройки и 1 целочисленного элемента настройки. При выходе из приложения и повторном его запуске содержимое ***EditText***-ов должно сохранятся.

**Указания к выполнению задания:**

Для сохранения данных небольшого объема будем использовать настройки (или предпочтения пользователя) – это специального вида файлы. Каждая настройка может содержать несколько элементов данных, для каждого элемента данных хранится ключ (имя данного) и значение. Настройки также имеют имена. И имена настроек, и имена данных (ключи) – это строки, удовлетворяющие требованиям, предъявляемым к именам переменных – они должны содержать только латинские буквы, цифры и нижние подчеркивания и не должны начинаться с цифры. Сами данные могут быть различных типов. Схематически настройки можно представить следующим образом:

**Настройка\_m**

|  |  |
| --- | --- |
| Ключ\_1 | Значение\_1 |
| Ключ\_2 | Значение\_2 |
| . . . | . . . |
| Ключ\_m1 | Значение\_m1 |

**Настройка\_1**

|  |  |
| --- | --- |
| Ключ\_1 | Значение\_1 |
| Ключ\_2 | Значение\_2 |
| . . . | . . . |
| Ключ\_n1 | Значение\_n1 |

. . .

В нашей работе мы должны использовать 4 строковых ресурса для хранения одного имени настройки (у нас одна настройка) и трех ключей элементов настройки.

С настройками мы работаем в приложении с помощью объектов класса ***SharedPreferences***. Помещать данные в настройки можно с помощью объектов класса ***SharedPreferences.Editor***.

При закрытии активности данные должны сохраняться в файлах-настройках, при открытии активности данные должны восстанавливаться из файлов-настроек.

Как мы узнаем, что активность открывается или закрывается и где следует писать код, который должен в этот момент выполняться? Чтобы ответить на этот вопрос, давайте рассмотрим жизненный цикл активности в Android.

Когда активность создается, происходит событие ***onCreate***. В этот момент производятся всевозможные начальные установки для активности. После события ***onCreate*** активность пока еще не видна. Для того, чтобы активность стала отображаться в окне (но, возможно, не на переднем плане и без возможности взаимодействия с пользователем), должно произойти событие ***onStart***. На передний план активность выходит и может после этого взаимодействовать с пользователем в результате события ***onResume***. «Противоположенное» для ***onResume*** событие – ***onPause*** – после него активность уходит с переднего плана, но остается видимой. ***onStop*** – событие, «противоположенное» ***onStart*** – после него активность перестает быть видимой, и, наконец, при события ***onDestroy*** активность разрушается, освобождаются все используемые ей ресурсы.

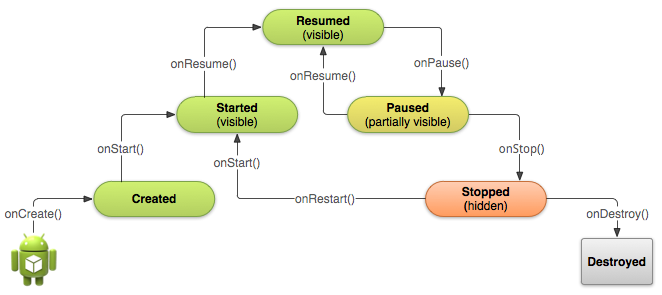


Схема взята с ресурса <http://developer.alexanderklimov.ru/android/theory/lifecycle.php>

Основной вывод, который мы можем отсюда сделать – это то, что при уходе активности с переднего плана (может быть, с последующим ее скрытием и уничтожением, а может быть и нет) обязательно происходит событие ***onPause***, следовательно, обработчик этого события - это то самое место, где разумно сохранить настройки. С другой стороны, при выходе активности на передний план всегда происходит событие ***onResume***, следовательно, обработчик этого события – это то самое место, где разумно восстановить настройки. В нашей работе настройка будет только одна, а в ней – 3 элемента данных.

public class MyActivity extends AppCompatActivity{

private SharedPreferences preferences; *//ссылка на объект-настройку*

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

preferences = getSharedPreferences( *//метод* ***getSharedPreferences*** *возвращает объект-настройку*

*//с именем, которое содержится в первом параметре.*

getString(R.string.preferences), *//имя настройки здесь берется из строкового ресурса*

MODE\_PRIVATE //*скрытый режим – только наше приложение может читать*

); //*этот файл-настройку*

}

@Override

protected void onResume() { //*обработчик события, которое помещает активность на передний план*

super.onResume(); *//вызов обработчика базового класса*

String intElKey = getString(R.string.int\_element); *//получаем из строкового ресурса ключ*

*//все ключи – строкового типа*

Int intEl = preferences.getInt(intElKey, 5); *//извлекаем из настройки целочисленное значение*

*//по его ключу, находящемуся в строке* ***intElKey****,*

*//если элемента с таким ключом в настройке нет,*

*//используем значение по умолчанию 5*

*//можно было предварительно проверить,*

*//существует ли в настройке элемент с таким ключом:*

*//* ***if(preferences.contains(intElKey)) …***

… *//извлекаем из настройки остальные элементы. Для строковых элементов используем* ***getString***

… *//извлеченные из настройки элементы (или значения по умолчанию) помещаем в* ***EditText****-ы*

}

@Override

protected void onPause() {  *//обработчик события, которое убирает активность с переднего плана*

super.onPause(); *//вызов обработчика базового класса*

… *//извлекаем сохраняемые значения из* ***EditText****-ов*

SharedPreferences.Editor editor = preferences.edit(); *//с помощью метода* ***edit*** *объекта-активности*

*//получаем объект-редактор* ***editor***

editor.putInt(getString(R.string.int\_element), value); *//с помощью редактора помещаем в настройку*

*//элемент. Первый параметр* ***putInt*** *– ключ //элемента, взятый из строкового ресурса,*

*//второй параметр – значение элемента,*

*//извлеченное из соответствующего* ***EditText****-а*

… *//аналогично помещаем в настройку остальные элементы. Для строковых элементов*

*//используем* ***putString***

editor.commit(); /*/применяем изменения настройки*

}

}

Преподаватель Яценко А.И.

(подпись)  
«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **7**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Разработка собственного адаптера и использование его в приложении. Знакомство с методом inflate класса LayoutInflater**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

# Проект 10. Свой адаптер

**Цель работы:**

* Создание своего адаптера
* Знакомство с методом ***inflate*** класса ***LayoutInflater***

**Задание:**

На главной активности расположен ***ListView***. Строковые элементы для ***ListView*** берутся из массива, который хранится в ресурсах. Фон для некоторых элементов подкрашен. Используем не менее двух цветов для подкраски фона, например, элементы с четными индексами подкрашиваются в красный цвет, а с нечетными – в синий.

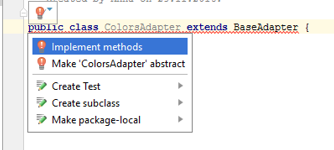
Внешний вид активности



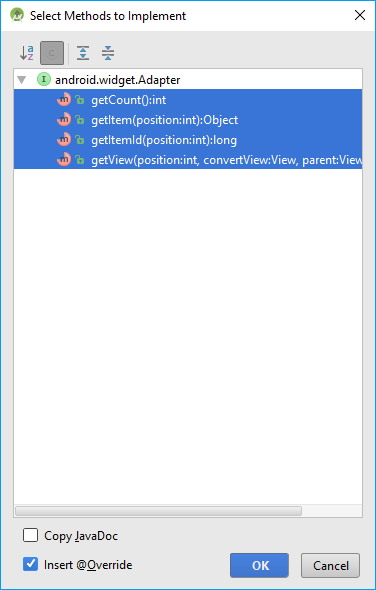
**Указания к выполнению задания:**

Здесь у нас каждый элемент списка оформлен по-своему, поэтому, вместо того, чтобы использовать стандартный адаптер, как мы это делали ранее, мы будем писать свой адаптер. Писать мы его будем не с чистого листа, а с использованием класса-базового адаптера как основы.

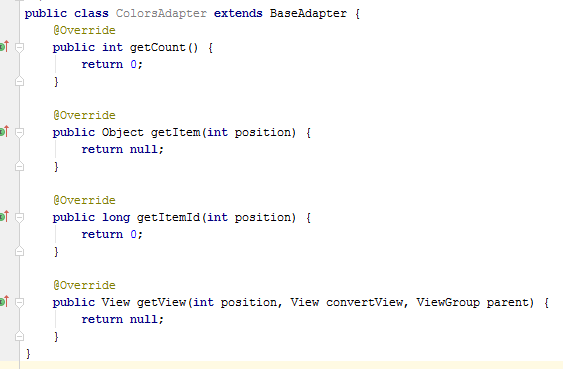
Создайте новый класс, не связанный с активностью. Сделайте его наследником от базового класса-адаптера BaseAdapter. Вам будет предложено реализовать методы этого класса.



Реализуйте все 4 предложенных метода.



У Вас получится следующая заготовка:



Класс-адаптер отвечает за то, чтобы элементы списка правильно отображались в списке. Класс умеет это делать, но не знает, например, сколько элементов в нашем списке или как должен выглядеть элемент нашего списка, он может получить эту нужную информацию только от нас с помощью вызова наших методов в нужные моменты времени. Аналогично вызывается, например, наш обработчик ***onClick*** в нужный момент времени из написанного не нами кода.



Мы тоже можем, конечно, вызывать в нашем коде указанные 4 метода.

Напишите теперь полностью нашу часть кода для класса:

public class ColorsAdapter extends BaseAdapter { *//наш класс (имя класса произвольное) является*

*//наследником от базового класса-адаптера* ***BaseAdapter****, а не от активности, как это было ранее*

private String[] names; *// нашу модель данных – массив строк для вывода в элементы*

*//списка – будем хранить в поле класса-адаптере*

*. . . /\* здесь опишите конструктор класса, который принимает один параметр – строковый массив*

*и присваивает этот параметр полю класса. Мы передаем адаптеру уже готовый массив, а не восстанавливаем его из ресурса прямо в адаптере, так как у класса-адаптера нет непосредственного доступа к ресурсам. Такой доступ имеется у класса – активности, поэтому мы восстановим массив из ресурса в классе-активности, а в адаптер передадим уже готовый массив. Из класса-адаптера тоже можно обращаться к ресурсам, но это менее удобно, чем из класса-активности \*/*

@Override

public int getCount() {

return names.length; *//возвращаем длину массива строк*

}

@Override

public Object getItem(int position) {

return names[position]; *//возвращаем элемент списка по его позиции в списке, в нашем случае*

*//элементом списка является строка*

}

@Override

public long getItemId(int position) {

return position; *//возвращаем идентификатор элемента списка, в данном случае считаем, что*

*//идентификатор совпадает с позицией элемента*

}

@Override

*/\* Этот метод формирует объект класса* ***View****, соответствующий элементу разметки, который*

*используется для отображения элемента списка. Метод вызывается для каждого элемента. При вызове нам известны параметры:*

*-* ***position*** *– на какой позиции в списке находится элемент, который в данный момент отрисовывается;*

***- view*** *– ссылка на объект-элемент разметки для текущего элемента списка. Возможно, что мы этот объект уже формировали, и он еще пока хранится, тогда его не требуется заново пересоздавать. Если ссылка пустая, то объект требуется создать;*

***- parent*** *– ссылка на список – родительский объект для элемента списка*

public View getView(int position, View view, ViewGroup parent) {

Context context = parent.getContext(); */\** ***Context*** *– это класс, который содержит многие полезные методы. Любая активность является наследником от этого класса, поэтому к его методам можно обращаться из класса-активности, но адаптер не является наследником от* ***Context****, следовательно, если нам потребуются методы* ***Context****-а в адаптере, следует получить там каким-либо образом объект класса* ***Context****, например, с помощью метода* ***getContext*** *списка. \*/*

if(view == null) { *//объекта – элемента разметки для нашего пункта списка не существует, поэтому*

*//мы его создаем*

LayoutInflater inflater = (LayoutInflater)

context.getSystemService(Context.LAYOUT\_INFLATER\_SERVICE); *//с помощью метода*

*//* ***getSystemService*** *объекта context класса* ***Context*** *создаем объект* ***inflater*** *класса* ***LayoutInflater***

view = inflater.inflate(R.layout.list\_item, parent, false); *//с помощью метода inflate объекта* ***inflate****r*

*//создаем из разметки для элемента списка, находящегося в файле* ***list\_item.xml*** *папки* ***layout****,*

*// дерево объектов и возвращаем ссылку на корневой элемент этого дерева. В данном случае*

*//дерево у нас состоит из одного узла - элемента* ***TextView***.

}

TextView textView = (TextView) view; *//преобразуем класс объекта – элемента разметки к* ***TextView***

textView.setText(names[position]); *//в элемент, находящийся в списке на позиции position, помещаем*

*//текст из элемента массива строк с индексом* ***position***

*… //устанавливаем цвет в зависимости от* ***position***

*//если мы хотим использовать цвет из ресурса цветов, то обращение к ресурсу может*

*//выглядеть так:*

*//****textView.setBackgroundColor(ContextCompat.getColor(context, R.color.имя\_ресурса));***

*//обратите внимание, что первый параметр метода* ***getColor*** *– это объект класса* ***Context****;*

*//когда мы использовали этот метод из активности, мы могли в качестве такого параметра*

*//указать* ***this****, так как активность – наследник от* ***Context****, следовательно, ее саму можно*

*//передать туда, где требуется* ***Context***

return view; *//возвращаем сформированный объект-элемент разметки*

}

}

Несколько слов о методе ***inflate***: этот метод из хранящегося в ресурсах файла-разметки xml создает аналогичной дерево программных объектов и возвращает ссылку на его корневой элемент. Далее по этой ссылке и по ***id*** нужного элемента разметки можно получить ссылку уже на нужный нам объект дерева с помощью метода ***findViewById***. Метод ***inflate*** вызывается в методе ***setContentView***, там создается дерево объектов из разметки, связанной с активностью. Схематически это можно изобразить так:

<LinearLayout

**android:id="@+id/ll"**

**…>**

<**ListView  
 android:id="@+id/lw"**

**…/>**

<LinearLayout

**android:id="@+id/ll2"**

**…>**

<Button

**android:id="@+id/b1"**

**…/>**

<TextView

**android:id="@+id/tv"**

**…/>**

</LinearLayout>

</LinearLayout>

LinearLayout ll

**inflate**

LinearLayout ll2

ListView lw

TextView tv

Button b1

Соответствующее дерево объектов

Пример файла-разметки

Слово ***inflate*** созвучно слову «инфляция» - накачивание, надувание, раздувание денежной массы, процесс превращения из плоского состояния в объемное. Будем считать, что разметка с тегами – это плоское состояние, а дерево программных объектов – объемное, соответственно, метод ***inflate*** «надувает», делает «объемной» нашу разметку.

В нашем случае файл разметки содержит единственный элемент – ***TextView***. Соответственно, дерево объектов будет состоять из одного узла.

Далее надо разместить в ресурсах массив строк с названиями элементов списка.

В методе ***onCreate*** класса – активности требуется восстановить массив строк из ресурса, создать объект класса нашего адаптера, передать ему в конструкторе этот массив строк и установить адаптер для списка ***ListView***.

. . . *//восстанавливаем массив строк из ресурса*

ColorsAdapter adapter = new ColorsAdapter(*имя\_массива\_строк*);

*имя\_объекта\_списка*.setAdapter(adapter);

Преподаватель Яценко А.И.

(подпись)  
«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **8**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Разработка проекта «Выбор цвета», включающего все освоенного в процессе выполнения предыдущих 10-ти проектов**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

# Проект 11. Выбор цвета

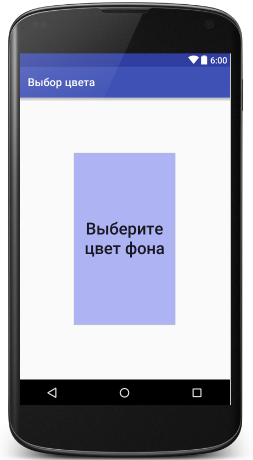
**Цель работы:**

* Создание относительно большого проекта на основе пройденного и освоенного в процессе создания предыдущих 10-ти проектов.

**Задание:**

На главной активности находится кнопка «Выберите цвет фона», по кнопке происходит переход на вторую активность со списком для выбора цвета фона. При щелчке по пункту списка может происходить (на Ваш выбор) либо сразу переход на главную активность, фон которой должен стать выбранным, либо установка выбранного цвета фона второй активности пока без перехода на главную. Во втором случае переход на главную активность должен осуществляться по кнопке «Back». В любом случае после выбора цвета фона он должен поменяться и у главной, и у второй активностей и сохраняться таким при переходах с главной на вторую активность и обратно до выбора следующего цвета фона. При закрытии приложения и повторном его запуске выбранный цвет фона должен сохраняться, причем краситься должны не сами активности, а окна. Коды и названия цветов должны храниться в массивах-ресурсах.

Вид главной и второй активностей:



**Указания к выполнению задания:**

Мы уже знаем практически все, что требуется для создания указанного проекта, за исключением двух пунктов:

* Установка цвета фона для окна (а не для активности, как в проекте 3):

***getWindow().getDecorView().setBackgroundColor(…);***

* Работа с целочисленным массивом-ресурсом:

Пример ресурса для целочисленного массива:

<array name="color\_values">

<item>#ffaaf6b3</item>

<item>#ffd0ffd0</item>

<item>#ff9adcff</item>

<item>#ffffffff</item>

</array>

Обратите внимание, что здесь хранятся 4, а не 3 составляющие цвета (***ARGB***, а не

***RGB***), первая составляющая – прозрачность – ff.

Восстанавливаем целочисленный массив из ресурса

***int[] values = getResources().getIntArray(R.array.color\_values);***

Далее можно отметить следующие моменты:

* В классе адаптере должны быть 2 массива – 1) целочисленный для хранения кодов цветов

2) строковый – для хранения названий цветов

Оба массива должны быть одинакового размера, цвета и коды должны друг другу соответствовать.

* Конструктор адаптера должен принимать 2 параметра – ссылки на целочисленный и строковый массивы - и присваивать параметры полям
* Сохранение целочисленного элемента настройки – кода цвета - происходит при щелчке по элементу списка
* Восстановление цвета из настройки происходит при событии ***onResume*** для главной активности

Примерный план выполнения проекта:

1. Создайте 2 активности с разметками и классами, список пусть пока будет пустым, убедитесь, что по кнопкам «***Выберите цвет фона***» и «***Back***» происходит, соответственно, переход на вторую активность и возврат обратно.
2. Создайте 2 массива-ресурса – один строковый с названиями цветов, а второй – целочисленный с кодами цветов. Массивы должны быть одинаковой длины, цвета и коды должны друг другу соответствовать. Создайте свой адаптер для списка, на уровне класса укажите 2 поля – строковый массив для названий цветов и целочисленный массив для кодов цветов, напишите конструктор, который принимает 2 параметра – массива и присваивает параметры полям. Напишите 4 метода для адаптера, метод ***getItem*** должен возвращать код цвета по его позиции, в методе ***getView*** текст для элемента списка должен браться из строкового массива, а цвет – из целочисленного массива. В классе – второй активности опишите поля для списка и адаптера, в методе ***onCreate*** восстановите оба массива из ресурса и передайте их адаптеру при его создании, свяжите список с адаптером. Запустите приложение, убедитесь, что на второй активности правильно выводятся элементы списка – цвета и их названия. Щелчок по элементу списка на этом этапе еще не обрабатывается.
3. Теперь в классе – второй активности напишите обработчик щелчка по элементу списка (см. проект 5). В обработчике известна позиция элемента списка, по которой щелкнули, требуется узнать код цвета, соответствующий этой позиции – это можно сделать с помощью метода ***getItem*** адаптера. Полученное целое значение – код цвета – требуется сохранить в настройке и, если вы реализуете версию проекта, в которой после щелчка пользователь остается на второй активности, ее окно надо перекрасить в выбранный цвет. Далее в событиях ***onResume*** и для главной и для второй активностей требуется прочитать настройку с кодом цвета и установить соответствующий цвет окна. Теперь приложение должно работать в соответствии с заданием.

Преподаватель Яценко А.И.

(подпись)  
«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **9**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Создание и использование своего базового класса-активности. Использование в приложении кнопки, доступность которой меняется в зависимости от содержимого EditText»

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

# Проект 12. Свой базовый класс - активность

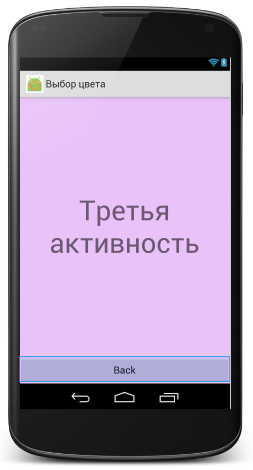
**Цель работы:**

* Создание своего базового класса-активности
* Распределение классов по разным пакетам

**Задание:**

Эта работа является продолжением проекта 11. В приложении могут открываться не 2, а 3 активности, каждая из них должна иметь выбранный цвет фона, код цвета фона восстанавливается из настройки. Код для восстановления цвета из настройки должен находиться в классе – базовой активности, все три активности должны быть наследниками от класса – базовой активности.

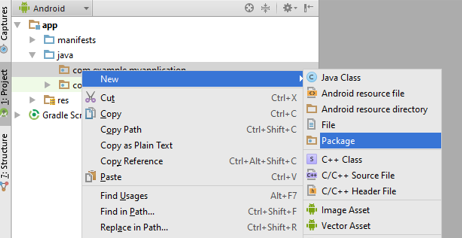
Вид активностей (выбран фиолетовый цвет фона):

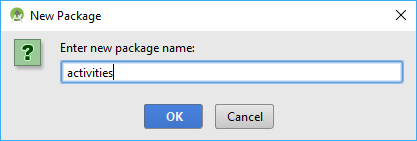


**Указания к выполнению задания:**

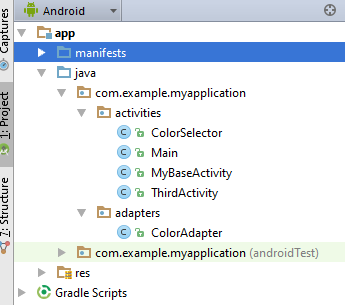
В данном проекте у нас будет 4 класса-активности, считая базовый, и 1 класс – адаптер. Не будем помещать все классы в один пакет/папку, а распределим их по нескольким пакетам/папкам, подобно тому, как мы упорядочиваем файлы на нашем компьютере по разным папкам, когда их становится достаточно много.

Далее указано, как можно создать вложенный пакет внутри главного:





В результате у нас должна получиться следующая структура:



Теперь обсудим код приложения.

Предположим, что мы создаем серьезное приложение, в котором имеется несколько десятков активностей, каждая из которых должна окрашиваться в выбранный цвет фона. Неудобно для каждой активности писать свой собственный обработчик ***onResume,*** в котором из настройки восстанавливается код цвета, поэтому давайте создадим базовый класс и сосредоточим в нем код, общий для всех наших активностей. Назовем его, например, ***MyBaseActivity*** – наследник от, допустим, ***AppCompatActivity***, а все наши активности сделаем наследниками от ***MyBaseActivity***. В ***MyBaseActivity*** перегрузите ***onResume*** и организуйте в нем чтение кода цвета из настройки и установку его как фонового цвета для окна.

Рекомендуется в ***MyBaseActivity*** определить ссылку на объект-настройку на уровне класса, создать объект-настройку в ***onCreate***, использовать настройку в ***onResume***. Не надо связывать класс ***MyBaseActivity*** ни с какой активностью – т.е. в ***onCreate*** для ***MyBaseActivity*** не надо вызывать метод ***setContentView***.

В классах – наследниках от ***MyBaseActivity*** не надо перегружать и вызывать ***onResume***, этот обработчик будет запускаться автоматически для каждой активности и при этом автоматически будет вызываться ***onResume*** для базового класса ***MyBaseActivity,*** который Вы уже перегрузили.



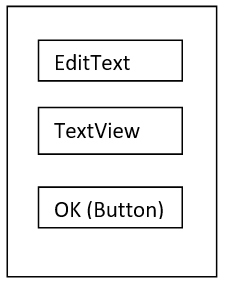
# Проект 13. Управление доступностью кнопки

**Цель работы:**

* Создание кнопки, доступность которой меняется в зависимости от содержимого EditText

**Задание:**

Вид активности:



Кнопка должна быть доступна только если в ***EditText*** есть текст (пробелы не считаются текстом). По кнопке содержимое ***EditText*** должно копироваться в ***TextView***.

**Указания к выполнению задания:**

Для того, чтобы следить за событиями, связанными с изменением текста в TextView, в главной активности реализуйте интерфейс TextWatcher, потребуется реализовать следующие методы:

@Override

*//это событие возникает перед изменением текста*

public void beforeTextChanged(CharSequence s, int start, int count, int after) {

*//можно оставить пустым*

}

@Override

*//это событие возникает при изменении текста*

public void onTextChanged(CharSequence s, int start, int before, int count) {

*… /\* здесь устанавливаем доступность кнопки в зависимости от содержимого EditText*

***имя\_объекта\_кнопки.setEnabled(true)***

*или*

***имя\_объекта\_кнопки.setEnabled(false)***

Для анализа строки можно использовать методы ***trim*** и ***length***,

метод ***trim()***, примененный к строке, убирает пробелы слева и справа и возвращает измененную

строку, метод ***length()*** возвращает длину строки.

*Вместо содержимого* ***EditText*** *можно анализировать значение параметра* ***CharSequence s****,*

*но, чтобы работать потом с ним как со строкой, его придется преобразовать к строковому*

*типу*

*\*/*

}

@Override

*//это событие возникает после изменения текста*

public void afterTextChanged(Editable s) {

*//можно оставить пустым*

}

Не забудьте установить обработчик изменения содержимого для ***EditText***, применив для него метод ***addTextChangedListener(this).***

Преподаватель Яценко А.И.

(подпись)  
«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **10**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Использование в приложении передачи данных между активностями с помощью объекта класса Intent**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

# Проект 14. Простейший обмен данными между активностями с использованием Intent

**Цель работы:**

* Организовать обмен данными между активностями с использованием объекта класса ***Intent***.

**Задание:**

Вид двух активностей:

Вторая активность

Главная активность

EditText

EditText

EditText

EditText

Вызов второй активности (Button)

OK (Button)

Cancel (Button)

При вызове второй активности содержимое элементов ***EditText*** должно ей передаваться и отображаться в ее собственных элементах ***EditText***.

При нажатии на ***OK*** второй активности происходит возврат к главной активности, содержимое элементов ***EditText*** второй активности (возможно, измененное) должно передаваться главной активности и отображаться там.

При нажатии на ***Cancel*** второй активности происходит возврат к главной активности, но содержимое ***EditText*** ей не передается и остается таким, каким оно было до вызова второй активности.

**Указания к выполнению задания:**

Объект класса ***Intent*** может использоваться не только для вызова одной активности из другой, но и для передачи данных между активностями. Для этого в ***Intent*** есть специальное поле – объект класса ***Bundle***:

Bundle

|  |  |
| --- | --- |
| Ключ1 | Значение 1 |
| Ключ2 | Значение 2 |
| … | … |
| Ключ n | Значение n |

Intent

В объект класса ***Bundle*** можно поместить несколько элементов данных различных типов – каждое со своим ключом. Все ключи всегда имеют строковый тип. В этом смысле ***Bundle*** похож на ассоциативный массив или на настройку (***SharedPreferences***) или на запись (***record***) в ***Pascal***-е или на структуру (***struct***) в ***c***-подобных языках, но с немного другой организацией доступа.

Схема обмена данными между активностями с помощью объекта класса ***Intent***:

Главная активность

***startActivityForResult***

(метод)

***onActivityResult***

(обработчик события)

Вторая активность

***onCreate***

(обработчик события)

***setResult***

***finish***

(методы)

***Intent*** с данными

***Intent*** с данными

Если мы хотим запустить активность для получения результата, мы используем метод ***startActivityForResult*** (вместо используемого ранее ***startActivity***). Далее мы попадаем в обработчик события ***onCreate*** для второй активности, там мы можем анализировать данные, переданные в вызываемую активность. После формирования результата вызываемой активностью и возврата в главную с помощью методов ***setResult*** и ***finish*** для главной активности возникает событие ***onActivityResult,*** в обработчике которого можно анализировать полученные результаты.

Одну и ту же активность можно вызвать с различными целями для формирования различных результатов, поэтому используется специальный параметр – идентификатор типа запроса на открытие активности.

Теперь можно перейти к коду:

*//На уровне класса объявляем константу (числовое значение - любое) – идентификатор типа //запроса на открытие активности, в данной работе у нас используется только один тип //запроса*

protected static final int MY\_ACTION=0x000314;

*//При вызове второй активности:*

Intent intent=new Intent(this, DataTest.class); *//создаем Intent*

intent.putExtra("str",editText1.getText().toString()); */\*передаем объекту intent содержимое* ***EditText1****, в данном примере ключ для содержимого – “****str****”, у вас при указании ключа должна использоваться ссылка на строковый ресурс. \*/*

*/\* Передайте intent-у содержимое* ***EditText2****, для этого содержимого используйте другой ключ (используйте ссылку на строковый ресурс) \*/*

startActivityForResult(intent, MY\_ACTION*); //запускаем активность для получения результата*

*/\*в зависимости от второго параметра - типа запроса на открытие второй активности – по-разному должен формироваться результат, пока мы используем только один тип запроса*

*Событие* ***onActivityResult*** *возникает при получении результата от вызванной активности \*/*

@Override

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

if(resultCode==RESULT\_OK){ *//если не было отмены*

Bundle extras=data.getExtras(); *//получаем ссылку на объект, содержащий данные*

extras.getString("str") *//извлекаем данные (здесь - строку), хранящиеся с ключом “****str****”, эту строку надо скопировать в* ***EditText.*** *Вы должны здесь использовать строковый ресурс*

*//вторую строку требуется извлечь и скопировать во второй* ***EditText****, использовать другой ключ*

*/\*при получении результата и его анализе может понадобиться информация о типе запроса, переданная через Intent при запуске второй активности и возвращенная обратно второй активностью через этот же Intent (доступ к типу запроса происходит через параметр* ***requestCode****). В данном примере мы используем только один тип запроса и параметр* ***requestCode*** *не анализируем.\*/*

}

}

*//Фрагмент из* ***onCreate*** *для 2 активности:*

@Override

protected void onCreate(Bundle savedInstanceState) {

// …

Bundle extras=getIntent().getExtras(); *//получаем* ***Intent****, с помощью которого была запущена вторая активность и далее – объект класса* ***Bundle****, который может содержать данные*

if(extras!=null){ *//проверяем, были ли переданы данные*

String txt=extras.getString("str");

if(txt!=null*) //проверяем, был ли передан элемент данных с ключом «****str****». Здесь Вы должны использовать строковый ресурс.*

*//помещаем извлеченную строку в* ***EditText****, тоже самое надо сделать и со 2 строкой*

}

}

*//Кнопка* ***ОК*** *для 2 активности:*

public void ready(View view){

Intent intent=getIntent(); */\*получаем* ***Intent****, с помощью которого была запущена вторая активность\*/*

intent.putExtra("str",/\* *данные из* ***EditText*** \*/); *//помещаем в объект класса Bundle*

*//полученного объекта Intent данные из соответствующего* ***EditText***

*//****”str”*** *– ключ для элемента данных*

*//Вам нужно будет поместить в тот же Intent еще один элемент данных из второго*

*//****EditText***

***//****Оба ключа должны браться из строковых ресурсов****.***

setResult(RESULT\_OK,intent);

finish();

}

*//Кнопка* ***Cancel*** *для 2 активности:*

public void onCancel(View view){

setResult(RESULT\_CANCELED);

finish();

}

Преподаватель Яценко А.И.

(подпись)  
«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **11**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Разработка приложения – записной книжки с использованием передачи данных между активностями с помощью объекта класса Intent**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

Проект 15. Разработка приложения – записной книжки с использованием передачи данных между активностями с помощью объекта класса Intent

Главная активность:

ListView

Add

Вторая активность:

EditText

Cancel

OK

EditText - многострочный (android:inputType="textMultiLine")

Кнопка OK доступна только если в EditText есть содержимое

Сделаем базовый класс для определения там констант CREATE\_ACTION, EDIT\_ACTION и ключей для передачи элементов данных (используем константы вместо использования ресурсов).

public class MyBaseActivity extends AppCompatActivity{

protected static final int CREATE\_ACTION = 0x000312;

protected static final int EDIT\_ACTION = 0x000313;

protected static final String EXTRA\_TEXT = "text";

protected static final String EXTRA\_ID = "id";

}

Эти константы – общие для всех классов-наследников и доступны во всех классах-наследниках

Наши активности – наследники от MyBaseActivity

При нажатии на кнопку Add должен происходить переход на вторую активность для добавления новой записи (1 тип запроса на вызов 2 активности, назовем его CREATE\_ACTION), при этом второй активности никаких данных не передается

При нажатии на элемент списка должен происходить переход на вторую активность для редактирования указанной записи (2 тип запроса на вызов 2 активности, назовем его EDIT\_ACTION), при этом второй активности должны передаваться два элемента данных – 1) целая позиция строки, по которой щелкнули и 2) сама строка. Строка должна отобразиться в EditText второй активности.

Вторая активность в любом случае возвращает (передает) в главную активность строку из EditText, если активность была вызвана для редактирования, то она передает еще и целочисленную позицию строки. Собственно, эта позиция и так находится в объекте Intent, полученном второй активностью от главной, вторая активность отсылает этот объект обратно первой активности с неизмененным значением позиции и, скорее всего, измененной самой строкой.

При получении результата главной активностью:

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

if(resultCode == RESULT\_OK) {

//извлекаем строку

switch (requestCode) {

case CREATE\_ACTION:

//добавляем строку в список

break;

case EDIT\_ACTION:

//извлекаем позицию строки

//редактируем в списке строку с данной позицией

break;

}

}

Преподаватель Яценко А.И.

(подпись)  
«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **12**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Использование уведомлений Toast. Создание собственной разметки для Toast.

Использование собственного класса-приложения для передачи данных между активностями»

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

Проект 16. Использование уведомлений Toast. Создание собственной разметки для Toast.

<http://developer.alexanderklimov.ru/android/toast.php>

Обратите внимание, что на странице есть опечатки, например, у тега TextView в файле-разметке отсутствует закрывающий слеш (/), так что копируем вдумчиво и творчески.

На главной активности – 6 кнопок и 2 EditText. Первые три кнопки работают так, как указано в первых трех примерах на странице по ссылке.

Четвертая кнопка примерно соответствует заданию «Создание собственных всплывающих уведомлений», но в разметку для Toast надо поместить ImageView и два элемента TextView, указать для них разные цвета текста и фона. Укажите для ImageView имя файла-изображения прямо в разметке, используя параметр src, укажите для обоих TextView тексты сообщений прямо в разметке.

!!! Вторым параметром для метода inflate можно указать null.

По пятой кнопке должно выводиться уведомление, соответствующее разметке, используемой четвертой кнопкой (д.б. только одна разметка), но изображение, тексты надписей и цвета должны быть другими, их следует установить программно:

**TextView tv=(TextView)layout.findViewById(R.id.*textView*);**

**//по ранее полученной ссылке на корневой элемент разметки (layout) получаем ссылку на какой-либо элемент (здесь на TextView) внутри корневого элемента, меняем (устанавливаем) у него текст и цвет.**

**Получите также ссылку на ImageView, измените используемое изображение.**

**ImageView imageView=(ImageView)layout.findViewById(R.id.*imageView*);  
imageView.setImageResource(R.drawable.*itest2*); //itest2 – имя файла с изображением**

**Измените текст и цвет для обоих TextView и изображение перед показом (show) уведомления.**

**Шестая кнопка:**

**- создайте класс ToastHelper, в нем должно быть 2 метода – конструктор и show. На уровне класса укажите поле context - объект класса Context. Конструктор должен принимать параметр класса Context и присваивать параметр полю. Этот объект потребуется в операторах метода show**

public void show(String message1, String message2 ){

LayoutInflater inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT\_INFLATER\_SERVICE);

**View layout = inflater.inflate(R.layout.*custom\_layout*,null);**

Toast toast = new Toast(context);

//устанавливаем для уведомления 2 текста сообщения для обоих TextView, подобно тому, как это делалось для 5 кнопки. Оба текста передаются в метод show как строковые параметры.

В классе главной активности указываем поле – объект класса ToastHelper, в методе onCreate для главной активности создаем сам объект, передаем конструктору this в качестве параметра – контекста.

По 6 кнопке читаем содержимое 2-х EditText-ов, вызываем метод show для объекта класса ToastHelper, передаем ему содержимое 2-х EditText-ов в качестве параметров.

Проект 17. Использование класса-приложения для передачи данными между активностями

**Вид активностей и работа всего приложения – как в задании 10, но для обмена данными между активностями не используем Intent, а используем класс-приложение.**

**Создаем собственный класс – наследник от приложения:**

**public class MyApp extends Application {  
//определяем 2 строковых поля – они играют роль глобальных переменных, к которым можно обратиться из всех активностей.**

**public MyApp(){ //конструктор  
super();  
//строковым полям присваиваем начальные значения НЕ ИЗ ресурсов**

**}**

**Пишем public set-еры и get-еры для этих двух строковых полей**

**!!! Не забываем указать имя нашего класса-приложения в манифесте**

**<application**

**android:name=".MyApp"**

**…**

**Класс – главная активность**

**protected MyApp myApp; //на уровне класса определяем поле – ссылку на объект-приложение (этот объект – единственный, его создаем не мы, мы только получаем на него ссылку)**

**В onCreate (кроме обычных действий):**

**myApp= (MyApp)getApplicationContext(); //получаем ссылку на объект-приложение**

**Восстанавливаем два строковых значения из ресурсов, устанавливаем значения двух глобальных строковых переменных через set-ры, эти же значения отображаем в EditText-ах**

**По кнопке перехода на вторую активность копируем строковые значения из EditText-ов в глобальные переменные.**

**А) Для вызова активности используем startActivityForResult, затем в обработчике onActivityResult получаем измененные строковые значения с помощью get-теров.**

**Или**

**Б) Для вызова активности используем startActivity, затем в обработчике onResum получаем измененные строковые значения с помощью get-теров.**

**Класс-вторая активность**

**С объектом-приложением работаем также, как и в первой активности.**

**В onResume**

**Восстанавливаем строковые значения с помощью get-теров, записываем их в EtitText-ы.**

**По кнопке ОК**

**Сохраняем строковые значения из EditText-ов с помощью set-теров.**

**НИГДЕ не используем Intent для передачи данных, только для вызова второй активности из главной.**

Преподаватель Яценко А.И.

(подпись)  
«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **13**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Разработка приложения с использованием метода inflate**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

Проект 18. Разработка приложения с использованием метода inflate.

Используем одну активность с корневым элементом **LinearLayout. Активность содержит еще один внутренний LinearLayout, в который будут добавляться элементы, а также 2 EditText-а и 2 кнопки. Кроме разметки для активности делаем еще 2 разметки для элементов, которые будут добавляться во внутренний LinearLayout.**

**По первой кнопке во внутренний LinearLayout добавляется элемент TextView, для которого мы создаем разметку с корневым элементом TextView, этот элемент в этой разметке – единственный. Наполнение (содержимое) для этого TextView берется из одного из EditText-ов.**

**LinearLayout lin= (LinearLayout) findViewById*(/\* указываем id-ресурс для ВНУТРЕННЕГО LinearLayout разметки для главной активности, в который предполагается добавлять элементы \*/*);  
 LayoutInflater inflater = (LayoutInflater)  
 getSystemService(Context.*LAYOUT\_INFLATER\_SERVICE*);  
 TextView textView = (TextView) inflater.inflate(*/\* указываем layout-ресурс добавляемого элемента TextView \*/*, null);  
  
 textView.setText(*/\* берем содержимое одного из EditText-ов \*/*);  
  
 lin.addView(textView); *//добавляем элемент во внутренний LinearLayout***

**По второй кнопке во внутренний LinearLayout добавляется более сложный элемент, который выглядит следующим образом:**

**Name: Andrey**

**Age: 19**

**Имя и возраст берем из EditText-ов**

Сделайте для этого элемента соответствующую разметку с корневым элементом **LinearLayout, внутри которого могут находиться другие элементы LinearLayout и элементы TextView.**

**LinearLayout lin= (LinearLayout) findViewById(*/\* указываем id-ресурс для ВНУТРЕННЕГО LinearLayout разметки для главной активности, в который предполагается добавлять элементы \*/*);  
 LayoutInflater inflater = (LayoutInflater)  
 getSystemService(Context.*LAYOUT\_INFLATER\_SERVICE*);  
 LinearLayout root = (LinearLayout) inflater.inflate(*/\* указываем layout-ресурс добавляемого сложного элемента \*/*, null);  
 TextView textView = (TextView) root.findViewById(/\* *указываем id-ресурс для TextView, в который мы будем помещать имя из EditText \*/*);  
 textView.setText(*/\* имя из EditText \*/*);  
*//аналогично в другой TextView помещаем возраст из второго EditText-а* lin.addView(root);**

Преподаватель Яценко А.И.

(подпись)  
«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **14**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Усовершенствование приложения - записной книжки путем использования собственного адаптера**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

Проект 19. Усовершенствование приложения - записной книжки путем использования собственного адаптера.

Внешний вид ПОЧТИ такой же, как и в работе 15. Функционал такой же. Обмен данными между активностями происходит также, как и в работе 15 – через Intent. Также используется класс – наследник от активности для хранения глобальных констант. Не забудьте, что во второй активности кнопка должна быть доступна только если в EditText есть содержимое. Сделайте EditText многострочным: android:inputType="textMultiLine"

Вид главной активности:

ListView

Record 1

Record 2

Record 3

Record 4

Add

Разметка для элемента списка – вместо TextView – LinearLayout, внутри которого находятся 2 TextView, один из них – цветной квадратик, второй содержит текст записи.

Класс-приложение:

Данные хранятся не внутри адаптера (стандартного или своего), а в классе-приложении. Данные хранятся в виде списка, к этому списку можно получить доступ из различных классов приложения:

private final List<String> notes = new ArrayList<>();

В конструкторе класса – приложения можно добавить в список несколько тестовых записей:

public MyApp(){

super();

notes.add("Record 1");

notes.add("Record 2");

}

В классе – приложении опишите get-тер для всего списка.

У такого списка есть методы:

* add(новая запись - строка) – добавление новой записи
* get(id) – возвращение одной записи – строки по ее id (не путать с возвращением всего списка)
* set(id, измененная запись-строка) – изменение строки с индексом id
* size() – возвращает длину списка (количество элементов в нем), это понадобится в адаптере

Главная активность

Определяем ссылку на объект-приложение (как мы это делали в работе 17).

Инициализируем эту ссылку с помощью метода getApplicationContext (как мы это делали в работе 17).

Определяем ссылку на List<String>. Инициализируем эту ссылку с помощью get-тера класса – приложения.

Далее везде, где мы раньше работали с данными через адаптер (adapter.add(…) или adapter.remove(…) или adapter.insert(…)) для добавления или изменения записи, работаем с вышеперечисленными методами списка List<String> Как и раньше, создаем объект для ListView, инициализируем его. !!! Не путайте список – модель данных List<String> и список - элемент ListView.

Для объекта ListView устанавливаем адаптер (как мы это делали ранее), но адаптер теперь у нас будет свой (мы уже писали свой адаптер в работах 10,11 и 12).

В обработчике onActivityResult в самом конце после добавления или изменения записи вызываем list.invalidateViews(); (list – объект ListView) для перерисовывания измененного списка.

В обработчике onItemClick содержимое изменяемой строки (то, что передается во 2-ю активность для изменения) определяем из модели данных (метод get для List<String>)

Класс-свой адаптер

На уровне класса можно описать поле Context context. Определить для класса конструктор с параметром Context, параметр присвоить полю. Также, как и в главной активности, в этом классе можно получить ссылку на объект-приложение, а потом – и на список-модель данных, но вызов метода getApplicationContext осуществляется через Context:

…= … context.getApplicationContext();

Перегружаем все 4 метода, которые будут предложены для перегрузки, везде, где работаем с данными, данные берем из модели List<String>. Например, там, где возвращается элемент, используем:

@Override

public String getItem(int position) {

return notes.get(position); //здесь notes – ссылка на строковый элемент списка из класса-приложения

}

В методе public View getView используем код, похожий на код из работы 7, дополненный с учетом работы 14, т.к. разметка для элемента списка здесь более сложная.

@Override

public View getView(int position, View view, ViewGroup parent) {

…

Преподаватель Яценко А.И.

(подпись)

«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **15**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Добавление вызова уведомлений Toast о внесении новой записи и редактировании существующей записи в приложение – записную книжку. Разработка и использование в приложении собственного элемента разметки.»

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

Проект 20. Добавление вызова уведомлений Toast о внесении новой записи и редактировании существующей записи в приложение – записную книжку.

Добавляем в предыдущее приложение вызов уведомлений о том, что запись изменилась или добавилась. Соответствующие тексты уведомлений храним в двух строковых ресурсах.

Определить класс ToastHelper (см. предыдущие работы), его метод show должен принимать 1 строковый параметр. Уведомление нестандартное, вид уведомления берется из Вашей собственной разметки.

Объект класса ToastHelper будет использоваться только в классе-приложении.

В классе-приложении определить методы-обертки с такими же параметрами для add и set класса List<String>. Имеется ввиду, что в методе add (или addNote) класса-приложения должен вызываться add класса List<String>. Метод add класса-приложения должен принимать тот же параметр, что и метод add класса List<String>. Кроме этого вызова в обертке должен вызываться метод show класса ToastHelper, в качестве параметра должна передаваться строка-сообщение о том, что запись добавлена. Аналогично сделать обертку для метода set.

Проект 21. Разработка и использование в приложении собственного элемента разметки.

а)В предыдущем приложении устанавливаем случайный цвет у цветного квадратика

б)Делаем простейший собственный элемент разметки и добавляем его в приложение

В классе-адаптере определяем ссылку на объект-генератор случайных чисел:

private final Random random;

В конструктере адаптера создаем объект класса Random, конструктору Random параметров не передаем.

В классе-адаптере определяем еще один метод, который возвращает случайный цвет (целочисленное значение). Этот метод не принимает параметров.

В методе используем

* вызов генератора случайных чисел: random.nextInt(n) – этот вызов возвращает случайное целое число в диапазоне от 0 до n-1
* метод Color.rgb(r,g,b), который принимает 3 составляющие цвета (от 0 до 255) и возвращает целочисленный код цвета.

В реализации метода getView адаптера устанавливаем случайный цвет у цветного квадратика с помощью вышеописанного метода.

Создаем свой собственный новый элемент разметки:

Определяем класс ColoredView – наследник от View. Android Studio предложит перегрузить четыре конструктора для нашего нового класса, перегружаем их все, кроме последнего. Определяем новый метод нашего класса – void setColor(int color), в методе устанавливается цвет фона элемента.

В разметке для элемента списка изменяем первый TextVew, соответствующий цветному квадратику, на наш элемент: начинаем писать <ColoredView , далее Android Studio нам поможет. Для нашего элемента оставьте установку квадратных размеров.

В реализации метода getView адаптера вместо устанавки случайного цвета для TextView устанавливаем случайный цвет для ColoredView, используем метод setColor

Преподаватель Яценко А.И.

(подпись)  
«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **16**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Использование уведомлений Notification. Создание собственной разметки для Notification. Добавление вызова уведомлений Notification о внесении новой записи и редактировании существующей записи в приложение – записную книжку. Определение глобальных статических констант в классе – приложении»

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

Проект 22. Использование уведомлений Notification. Создание собственной разметки для Notification.

На главной активности находятся 3 кнопки и 1 EditText. В классе определены 3 целые константы – идентификаторы уведомлений (Notification, а не Toast). По первым двум кнопкам отправляются уведомления, которым соответствуют две разные константы. У этих двух уведомлений –одинаковые иконки и они используют одинаковые разметки как для сообщения, так для открываемой активности. Для отправления этих двух уведомлений можно использовать один и тот же метод showNotification(int id, String text, вызываемый с разными параметрами. Кроме id уведомления этому методу в качестве параметра передается текст сообщения. Текст сообщения возьмите из EditText.

По третьей кнопке должно передаваться 3-е уведомление, для которого используются другие иконки, другая разметка для сообщения (пусть у нее будет другой внешний вид, напрмер, пусть изображение будет справа) и другая открываемая активность (пусть она выглядит по-другому, не так, как для уведомлений, отправляемых по двум первым кнопкам), соответственно для отправки уведомления понадобится другой метод, аналогичный showNotification.

------------- Пример кода ------------------

**private void showNotification(int id, String text){  
 Intent intent = new Intent(this, SecAct.class); //класс активности, которая открывается при просмотре сообщения об уведомлении. У Вас на этой активности может находиться, например, один TextView c текстом «Просмотр сообщения».  
 intent.setAction(Intent.*ACTION\_MAIN*);  
 intent.addCategory(Intent.*CATEGORY\_LAUNCHER*);  
 PendingIntent contentIntent = PendingIntent.*getActivity*(this, 0, intent, 0);  
  
 RemoteViews view = new RemoteViews(getPackageName(), R.layout.*notification*); //разметка для сообщения об уведомлении  
 view.setImageViewResource(R.id.*image*, R.mipmap.*ic\_launcher*);  
 view.setTextViewText(R.id.*title*, getString(R.string.*app\_name*));  
 view.setTextViewText(R.id.*description*, text);  
  
 NotificationCompat.Builder builder = new NotificationCompat.Builder(this);  
 builder  
 .setSmallIcon(R.mipmap.*ic\_launcher*)  
 .setContentTitle(getString(R.string.*app\_name*))  
 .setContentIntent(contentIntent)  
 .setAutoCancel(true)  
 .setTicker("Новое сообщение") //уберите эту настройку для 3-го уведомления  
 .setDefaults(NotificationCompat.*DEFAULT\_ALL*)  
 .setContent(view);  
 Notification notification = builder.build();  
  
 *//notification.flags |= Notification.FLAG\_NO\_CLEAR;* NotificationManager manager =  
 (NotificationManager) getSystemService(Context.*NOTIFICATION\_SERVICE*);  
 manager.notify(id, notification);//отправка уведомления  
}  
------ пример разметки для сообщения ----------  
*<?*xml version="1.0" encoding="utf-8"*?>*<RelativeLayout xmlns:android="**[**http://schemas.android.com/apk/res/android**](http://schemas.android.com/apk/res/android)**"  
  
android:layout\_width="match\_parent"  
android:layout\_height="wrap\_content">  
<ImageView  
 android:id="@+id/image"  
 android:layout\_marginRight="@dimen/padding"  
 android:layout\_alignParentLeft="true"  
 android:layout\_alignParentTop="true"  
 android:layout\_width="40dp"  
 android:layout\_height="40dp" />  
<TextView  
 android:id="@+id/title"  
 android:layout\_toRightOf="@id/image"  
 android:layout\_marginBottom="@dimen/padding"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content" />  
<TextView  
 android:id="@+id/description"  
 android:layout\_toRightOf="@id/image"  
 android:layout\_below="@id/title"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content" />  
</RelativeLayout>**

Проект 23. Добавление вызова уведомлений Notification о внесении новой записи и редактировании существующей записи в приложение – записную книжку. Определение глобальных статических констант в классе – приложении.

Добавьте отправку уведомлений о том, что запись добавлена или изменена, в предыдущее приложение (работа 17). Глобальные статические константы можно определить в классе-приложении, там же (или в отдельном специальном классе) определить метод **showNotification (нужен только один подобный метод, один вид иконок, одна разметка под сообщение)**. При просмотре сообщения должна открываться главная активность, где уже будет видна добавленная или измененная запись. В тексте уведомления должно указываться, добавляется запись или изменяется, и содержимое самой записи. **showNotification вызываем в тех же методах, что и show для Toast-ов.**

Преподаватель Яценко А.И.

(подпись)  
«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **17**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Разработка приложения с использованием списка – модели данных, в котором хранятся объекты пользовательского класса**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

Проект 24. Разработка приложения с использованием списка – модели данных, в котором хранятся объекты пользовательского класса.

Изменяем приложение (например, из работы 19), так, чтобы список-модель данных хранил не строки String, а объекты нашего класса. Пусть наш класс называется Note. Класс должен содержать не менее 2-х строковых полей (например, заголовок записи и содержание записи) и поле класса Time, в котором будет хранится время создания/изменения записи. Определяем для класса конструктор с тремя параметрами, который присваивает параметры полям. Определяем методы класса – set-еры и get-еры для всех элементов. Alt/Insert - получение помощи Android-Studio при определении вышеуказанных методов.

Заголовок класса Note:

public class Note implements Serializable {

…

Мы делаем наш класс Serializable для того, чтобы можно было передавать объект такого класса с помощью объекта Intent как единое целое, а не отдельно по полям.

Везде, где в классе-приложении используется тип String как тип элемента списка, надо String поменять на Note.

Обмен данными между активностями:

intent.putExtra – помещает в Intent отдельно id и отдельно объект класса Note, причем весь объект целиком, а не отдельно по полям

Извлечение из Intent объекта класса Note:

Note note = (Note) extras.getSerializable(EXTRA\_NOTE); //здесь EXTRA\_NOTE – ключ, под которым хранится в extras объект класса Note.

Получение текущего системного времени:

Time m\_time = new Time(System.currentTimeMillis());

Объект класса Time можно преобразовать в String с помощью метода toString()

Активность для редактирования

Разметку активности для редактирования надо изменить так, чтобы можно было бы добавлять и редактировать все поля (кроме Time)

Разметка для элемента списка ListView

Разметку для элемента списка надо изменить так, чтобы при выводе списка выводились все поля (см. работа N 14).

Свой адаптер

Класс-адаптер надо изменить так, чтобы при формировании view поля объекта попадали в соответствующие элементы разметки (см. работа N 14). Используйте get-теры класса Note для извлечения полей объекта.

Преподаватель Яценко А.И.

(подпись)

«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **18**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Использование файловой базы данных SQLite для создания и отображения таблицы. Использование файловой базы данных SQLite для поиска по таблице, изменения и удаления данных»

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

Проект 25. Использование файловой базы данных SQLite для создания и отображения таблицы.

На главной активности – 3 EditText, 1 ListView, 2 кнопки. Таблица БД состоит из 4 полей – ID, Фамилия (строка), Имя(строка), Возраст(целое). В примере таблица состоит из 3 полей, Вам надо ее немного переделать. По нажатии на одну из кнопок содержимое EditText-ов (Фамилия, Имя, Возраст) добавляется в новую строку таблицы. По нажатии 2-ой кнопки таблица отображается в ListView. Каждая строка таблицы отображается в одной строке ListView. Для ListView можно использовать стандартный строковый адаптер, строку для списка формируем путем объединения полей одной строки таблицы.

public class TestsHelper extends SQLiteOpenHelper {

private static final int DATABASE\_VERSION = 1;

private static final String DATABASE\_NAME = "testDB";

//внутренний класс

public static final class Tests {

public static final String TABLE\_NAME = "testTable";

public static final String COLUMN\_NAME = "name";

public static final String COLUMN\_AGE = "age";

public static String getCreateStatement(){

return String.format(

"CREATE TABLE %s(" +

"%s INTEGER PRIMARY KEY AUTOINCREMENT," +

"%s TEXT," +

"%s INTEGER" +

")",

TABLE\_NAME,

BaseColumns.\_ID,

COLUMN\_NAME,

COLUMN\_AGE

);

}

public static String getTest(Cursor cursor){

int ID = cursor.getColumnIndex(BaseColumns.\_ID),

nameID = cursor.getColumnIndex(COLUMN\_TITLE),

ageID = cursor.getColumnIndex(COLUMN\_DESCRIPTION);

testName=cursor.getString(nameID);

testAge=cursor.getString(ageID);

return(/\* вернуть строку, в которой объединены имя и возраст \*/);

}

public static long insertTest(SQLiteDatabase db, String testName, int testAge){

ContentValues values = new ContentValues();

values.put(COLUMN\_NAME,testName);

values.put(COLUMN\_AGE, testAge);

return db.insert(TABLE\_NAME, null, values);

}

}

public TestsHelper(Context context) {

super(context, DATABASE\_NAME, null, DATABASE\_VERSION);

}

@Override

public void onCreate(SQLiteDatabase db) {

db.execSQL(Tests.getCreateStatement());

}

@Override

public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

}

}

-----------------------------------------------------------------------------------------------------------

В главной активности(на уровне класса):

private TestsHelper dbHelper;

private Cursor cursor;

В OnCreate (главной активности):

dbHelper = new NotesHelper(this);

loadNotes();

public int getNotesCount(){

return cursor.getCount();

}

------------------- др. Методы главной активности --------------------------------

------- можно продумать перенос некоторых методов в TestsHelper --------------------------------------

public void loadTests(){

cursor = dbHelper.getReadableDatabase().rawQuery(

String.format(

"SELECT \* FROM %s",

TestsHelper.Notes.TABLE\_NAME

), null);

}

public String getTest(int id){

cursor.moveToPosition(id);

return TestsHelper.Tests.getNote(cursor);

}

public void addTest(String testName, int testAge){

NotesHelper.Notes.insertNote(

dbHelper.getWritableDatabase(),

testName, testAge

);

loadTests();

}

Проект 26. Использование файловой базы данных SQLite для поиска по таблице, изменения и удаления данных.

В предыдущую работу добавляем поиск по таблице, изменение записей таблицы и удаление записей таблицы, используем данные из EditText-ов. Не используем таблицу из примера. При выводе таблицы/результата запроса используем не переход на запись с определенным id, а переход на следующую запись (этот метод найдите в примере).

<http://startandroid.ru/ru/uroki/vse-uroki-spiskom/76-urok-36-sqlite-podrobnee-pro-metod-query-uslovie-sortirovka-gruppirovka.html>

<http://startandroid.ru/ru/uroki/vse-uroki-spiskom/75-urok-35-metody-query-i-delete-s-ukazaniem-uslovija.html>

Преподаватель Яценко А.И.

(подпись)  
«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **19**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Усовершенствование приложения - записной книжки путем сохранения ее в базе данных**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

Проект 27. Усовершенствование приложения - записной книжки путем сохранения ее в базе данных.

Изменяем работу 24 (или 33, если Вы уже ее выполнили) так, чтобы данные хранились не в списке List<MyClass>, а в базе данных. Методы, связанные с работой с БД и cursor переместите из активности в класс-приложение.

Преподаватель Яценко А.И.

(подпись)

«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **20**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Разработка приложения с использованием элемента Spinner. Использование в приложении класса - перечисления»

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

Проект 28. Разработка приложения с использованием элемента Spinner.

ListView

Spinner

Spinner

Add 2

Add 1

Разметка для spinner:

<Spinner

android:id="@+id/priority"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

**Работа со Spinner**

В onCreate для активности:

ArrayAdapter adapterForSpinner = new ArrayAdapter<>(this, разметка для элемента spinner – в данном случае TextView);

adapterForSpinner.addAll(массив строк);

объект-Spinner.setAdapter(adapterForSpinner);

объект-Spinner.setSelection(0);

В обработчике для кнопки:

Извлекаем индекс выбранного элемента:

объект-Spinner.getSelectedItemPosition()

**По-другому организовываем обработчики для кнопок:**

Для 1 кнопки:

В классе объявляем:

private View.OnClickListener forBtn1 =

new View.OnClickListener() {

@Override

public void onClick(View view) {

//код для обработчика 1

}

};

В onCreate:

Объект-кнопка1.setOnClickListener(forBtn1);

Для 2 кнопки:

Объект-кнопка2.setOnClickListener(

new View.OnClickListener() {

@Override

public void onClick(View view) {

//код для обработчика 2

}

}

);

Проект 29. Использование в приложении класса - перечисления

Перечисления (enum)

Пример перечисления:

public enum Priority { //храним в отдельном файле

High,

Medium,

Low;

public static Priority valueOf(int ordinal){ //возвращение объекта по его индексу

for (Priority item: values()){

if(item.ordinal() == ordinal){

return item;

}

}

throw new IllegalArgumentException();

}

}

Обращение к объекту-перечислению (пример): Priority.Low

Объявление, инициализация, присваивание: Priority pr=Priority.Medium;

Определение индекса: int code1=pr.ordinal();

Или int code2=Priority.High.ordinal();

Использование valueOf: Priority pr=Priority.valueOf(2); //вернется Priority.Low

Для enum существует метод value(), который возвращает массив объектов для перечисления

Этот метод можно использовать, например, для того, чтобы добавить все соответствующие перечислению объекты в Spinner:

ArrayAdapter adapter = new ArrayAdapter<>(this, /\* ресурс для элемента Spinner \*/);

adapter.addAll(Priority.values());

Работа:

Определите enum Sex для пола: male, female

Определите enum Status для статуса: pupil, student, worker, teacher

Определите класс Person с полями:

name (String), age(int), Sex, Status

Определите для этого класса конструктор с 4-мя полями, в котором параметры присваиваются полям.

Определите для этого класса метод toString(), выводящий поля объекта в строку

Add4

Add3

Add2

Add1

Status:

Sex:

Age:

ListView

Name:

Add5

Clear

Для Spinner создаем НЕ ArrayAdapter<String> adapter = new ArrayAdapter<>(this, /\* ресурс \*/);

а такой адаптер:

ArrayAdapter adapter = new ArrayAdapter<>(this, /\* ресурс \*/);

Add1 – добавляем в ListView индекс выбранного элемента Status (использовать getSelectedItemPosition для Spinner см. Предыдущую работу)

Add2 – добавляем в ListView сам выбранный Status (использовать getSelectedItemPosition и valueOf – восстанавливаем объект по индексу)

Add3 – добавляем в ListView массив объектов Sex

Add4 – добавляем в ListView массив объектов Status

Add5 – создаем объект класса Person, присваиваем его полям данные из соответствующих элементов EditText и Spinner и добавляем объект в ListView

Clear – очистка ListView

Преподаватель Яценко А.И.

(подпись)  
«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **21**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Усовершенствование приложения – записной книжки путем добавления в пользовательский класс поля – перечисления**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

Проект 30. Усовершенствование приложения – записной книжки путем добавления в пользовательский класс поля – перечисления.

Переделываем работу 24 или (лучше) 27 из «Расширенного списка работ» (им соответствуют работы 20 и 23 соответственно из файла «Практика»), добавляем в наш класс поле-перечисление (например, Priority), перечисление необходимо определить как Serializable (аналогично тому, как это сделано для класса Note). В ListView показываем значение этого поля вместе с другими полями, в активность для добавления/редактирования записи добавляем Spinner для работы с полем-перечислением. Не забудьте, что при редактировании записи первоначально в Spinner–е должно быть выбрано правильное значение. В базе данных перечисляемое поле хранится в виде целого числа – порядкового номера перечисления.

Преподаватель Яценко А.И.

(подпись)

«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **22**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Использование элементовDatePicker и TimePicker**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

Проект 31. Использование элементовDatePicker и TimePicker

<http://developer.alexanderklimov.ru/android/views/datepicker.php>

<http://developer.alexanderklimov.ru/android/views/timepicker.php>

TimePicker

Activity

Время прихода

15:40

Время ухода

17:20

установить время прихода

установить время ухода

Установить текущее время для TimePicker

TimePicker component

DatePicker

Activity

Дата

получения

книги

19.01.2016

Дата

возврата

книги

23.01.2016

установить дату получения книги

установить дату возврата книги

Установить текущую дату для DatePicker

DatePicker component

Main Activity

To DatePicker

Activity

To

TimePicker

Activity

Всего 3 активности (для всех элементов-кнопок указаны красные границы)

1. Main Activity содержит 2 кнопки, по которым происходит переход на 2 другие активности
2. DatePicker Activity содержит 3 TextView, 3 кнопки и компонент DatePicker (можете использовать вариант с календарем или без календаря, в разметке для этого указываем

android:calendarViewShown="false" или

android:calendarViewShown="true"

)

По нажатию на кнопку «Установить дату получения книги» выбранная в DatePicker дата устанавливается как дата получения книги, аналогично с датой возврата книги. По нажатию на кнопку «Установить текущую дату для DatePicker» для компонента DatePicker должна установиться текущая дата.

1. TimePicker Activity устроена аналогично DatePickerActivity, только она работает со временем.

Преподаватель Яценко А.И.

(подпись)

«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **23**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Использование диалогового окна AlertDialog**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

Проект 32. Использование диалогового окна AlertDialog

<http://developer.alexanderklimov.ru/android/alertdialog.php>

На главной активности – 5 кнопок и TextView. По кнопкам появляются диалоговые окна. В TextView отображается информация, полученная с помощью диалоговых окон.

По первой кнопке – пример из ссылки AlertDialog со списком

По второй кнопке - пример из ссылки AlertDialog с переключателями

По третьей кнопке - пример из ссылки AlertDialog с флажками

По четвертой кнопке – ввод данных для объекта класса Person с помощью диалогового окна

По пятой кнопке – ввод даты или времени с помощью диалогового окна

Для диалоговых окон, запускаемых по 4 и 5 кнопкам, делаем собственные разметки. Как можно получить данные из элементов разметки, можно понять из примера AlertDialog с рейтингом/решение проблемы/модифицированный код.

Преподаватель Яценко А.И.

(подпись)

«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **24**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Усовершенствование приложения – записной книжки путем использования элементов DatePicker и TimePicker и диалогового окна AlertDialog для установки времени и даты, являющихся полями пользовательского класса**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

Проект 33. Усовершенствование приложения – записной книжки путем использования элементов DatePicker и TimePicker и диалогового окна AlertDialog для установки времени и даты, являющихся полями пользовательского класса.

Переделываем работу 24 или (лучше) 27 из «Расширенного списка работ» (им соответствуют работы 20 и 23 соответственно из файла «Практика»), время для добавляемой/редактируемой записи берем не текущее, а устанавливаем его с помощью TimePicker, этот компонент располагаем в диалоговом окне, которое вызывается по кнопке из активности для добавления/редактирования.

Преподаватель Яценко А.И.

(подпись)

«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **25**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Разработка приложений с использованием меню**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

Проект 34. Разработка приложений с использованием меню.

<http://startandroid.ru/ru/uroki/vse-uroki-spiskom/45-urok-13-sozdanie-prostogo-menju.html>  
<http://startandroid.ru/ru/uroki/vse-uroki-spiskom/46-urok-14-menju-gruppy-porjadok.html>

Преподаватель Яценко А.И.

(подпись)

«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

(ФГБОУ ВПО «СПбГПУ»)

Университетский политехнический колледж

**Практическая работа №** **26**

по дисциплине **«Разработка программного обеспечения для мобильных устройств»**

**Тема: «**Разработка приложений с использованием перехватчиков широковещательных сообщений**»**

**Инструктаж по проведению практической работы**.

**Выполнение практической работы**: оказание оперативной помощи в процессе выполнения заданий практической работы, наблюдения, контроль выполнения студентами практической работы.

Проект 35. Разработка приложений с использованием перехватчиков широковещательных сообщений.

<http://startandroid.ru/ru/uroki/vse-uroki-spiskom/177-urok-107-android-3-actionbar-razmeschenie-elementov.html>

Преподаватель Яценко А.И.

(подпись)

«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г.