

ПРОГРАММИРОВАНИЕ МОБИЛЬНЫХ
ПРИЛОЖЕНИЙ ПОД ПЛАТФОРМУ

ANDROID



Урок №1

Введение в Android.
Основы Android.
Установка
необходимого ПО

Содержание

1.	Установка среды разработки Android Studio	5
1.1.	Выбор версии Android Studio	5
1.2.	Установка Android Studio	7
2.	Понятие SDK, SDK-Manager, Эмулятор, AVD	14
2.1.	SDK Manager	15
2.2.	Android Virtual Device Manager (AVD Manager)	20
2.3.	Создание виртуальных устройств для эмуляторов.....	21
3.	Создание проекта Android Studio	30
4.	Структура проекта разрабатываемого приложения в Android Studio	39
4.1.	Модули проекта	39
4.2.	Структура модуля проекта	40

4.3. Манифест приложения	43
4.4. Файл исходного кода класса Активности	44
4.5. Файл исходного кода разметки внешнего вида Активности	47
5. Написание первого приложения «Hello World»	52
6. Запуск приложения. Эмуляторы	59
6.1. Эмулятор Google-Android	59
6.2. Эмулятор Genymotion	62
6.3. Запуск приложения на реальном устройстве	69
7. Логи приложения. Класс android.util.Log	70
8. Активность. События, через которые проходит Активность	72
9. Верстка макета Активности с помощью XML	80
10. Базовые виджеты: TextView, Button, EditText	84
10.1. TextView	85
10.2. Button	86
10.3. EditText	89
11. События клика, перемещения при касании	90
12. Менеджеры раскладки LinearLayout, TableLayout	99
12.1. Понятие менеджера раскладки (Layout Manager)	99
12.2. LinearLayout	100

12.3. TableLayout	110
13. Программное создание элементов управления Button, TextView, EditText.....	117
14. Программное создание LinearLayout, TableLayout.....	120
15. Программная «верстка» макетов Активности...	123
16. Поворот экрана. Создание макета Активности для альбомной ориентации.....	130
17. Домашнее задание	135

1. Установка среды разработки Android Studio

1.1. Выбор версии Android Studio

Среда разработки Android Studio предназначена для разработки приложений для устройств, работающих под управлением операционной системы Android.

Android Studio является бесплатным программным продуктом. Она доступна для скачивания по ссылке <http://developer.android.com/sdk/index.html>. Безусловно, при установке вам необходимо будет согласиться с лицензионным соглашением, предлагаемым компанией Google.

Итак, заходим на страницу для скачивания, изучаем системные требования, находим необходимую версию для скачивания (см. Рисунок 1.1) в разделе «All Android Studio Packages».

Рекомендуемая версия Android Studio для операционной системы Windows включает в себя и некоторые компоненты последней версии SDK, поэтому ее размер превышает 1 Гб. Версии Android Studio для Linux и MacOS включают в себя только среду разработки и утилиты (AVD Manager, SDK Manager и так далее), поэтому после установки на операционные системы Linux/MacOS необходимо будет скачивать еще и SDK. В любом случае, загружать различные версии SDK рано или поздно все равно придется — так как компания Google постоянно выпускает новые версии SDK для новых версий ОС.

Android, поэтому для разработчиков мобильных приложений является обычным делом скачивание и установка новых версий SDK.

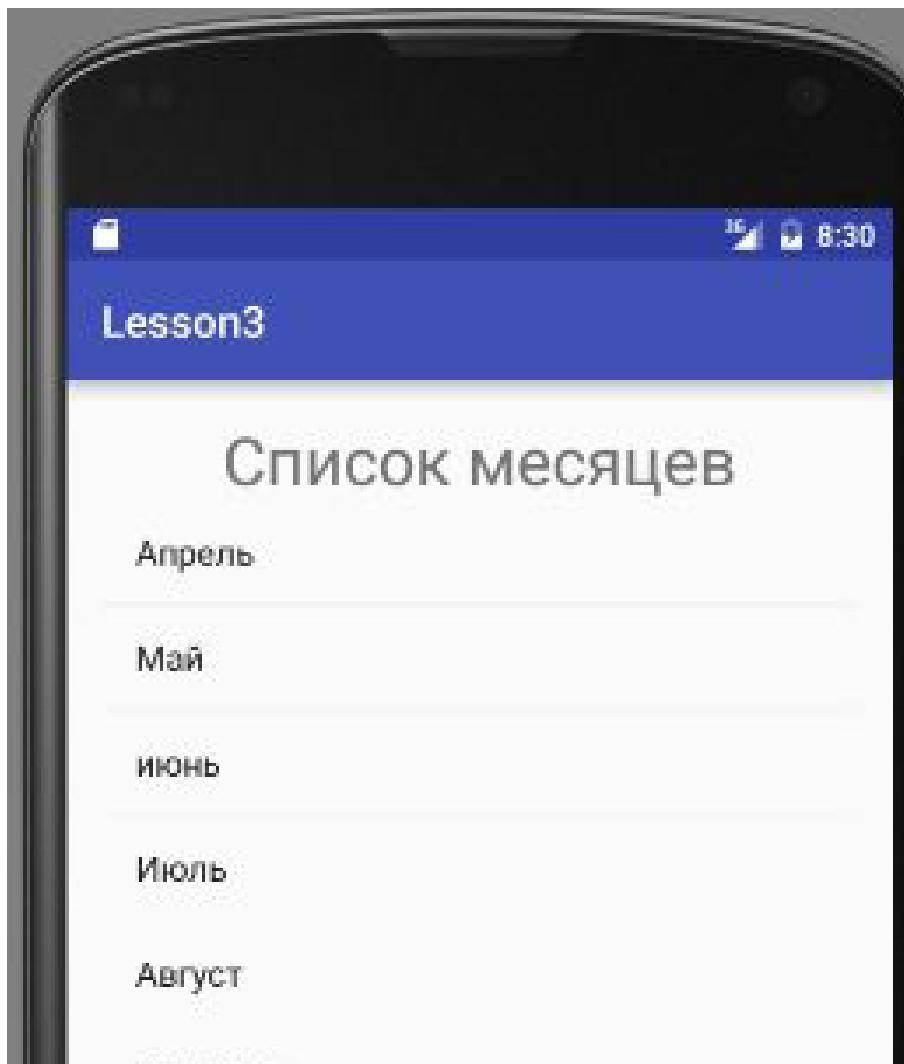


Рис. 1.1. Выбор версии Android Studio для скачивания

1.2. Установка Android Studio

Для успешной установки среды разработки Android Studio необходимо, чтобы на вашем компьютере была установлена платформа JDK (*Java Developer Kit*). Вам должна быть знакома эта платформа из курса «Программирование на языке Java», поэтому процесс установки JDK в этом уроке описываться не будет.

Запускайте установочный файл. Появляется окно приветствия (см. Рис. 1.2). Нажимайте на кнопку «Next». Появится следующее окно (Рис. 1.3), в котором вам будет предложен выбор — устанавливать Android Studio для всех пользователей данного компьютера или только для вас.

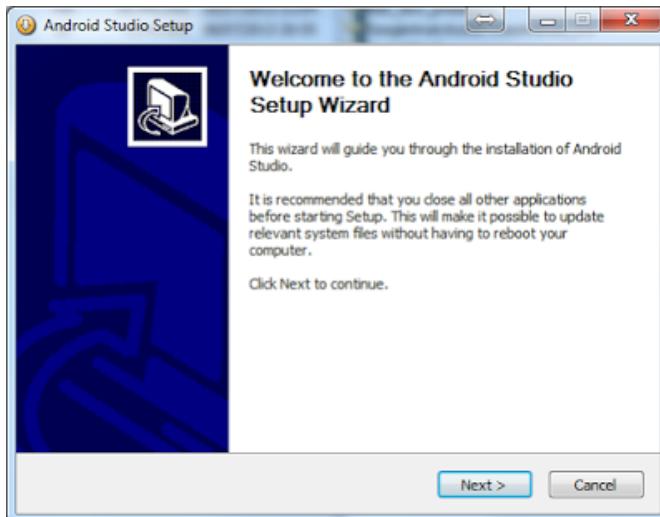


Рис. 1.2. Окно приветствия начала установки

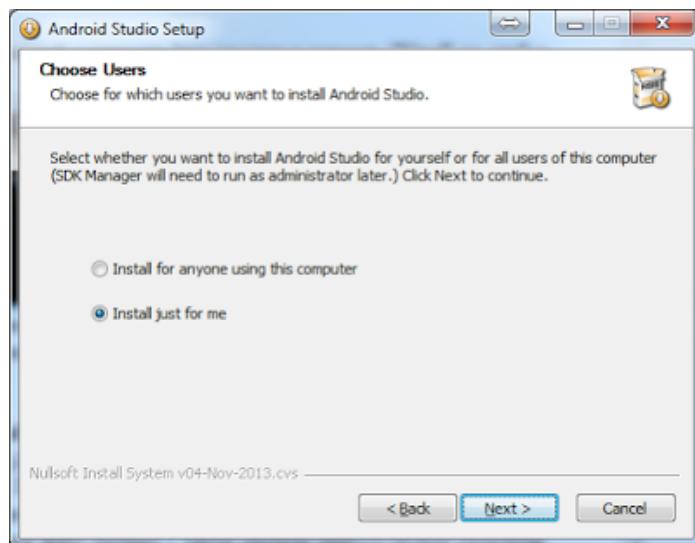


Рис. 1.3. Окно выбора доступности Android Studio другим пользователям компьютера

При выборе установки для всех пользователей, среда разработки Android Studio будет доступна для любого пользователя вашего компьютера, однако, каждому пользователю необходимо будет для себя лично скачивать и устанавливать необходимые версии SDK, которые будут располагаться в его личном профайле. Какой вариант выбрать — зависит от вашего решения и на процесс создания приложений это никак не повлияет. Делайте свой выбор и переходим к следующему окну. Если вы выбрали установку Android Studio только для себя, то появится окно (Рис. 1.4) в котором вам будет предложено выбрать каталог на для установки Android Studio. Прописывайте путь и наживайте кнопку «Next».

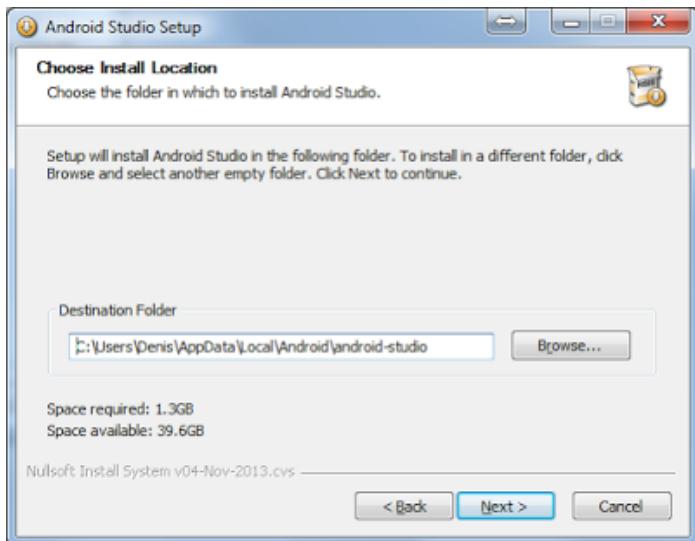


Рис. 1.4. Путь к каталогу для установки Android Studio

Следующее окно — выбор «Start Menu Folder» (Рис. 1.5). Действия в этом окне должны быть вам очень хорошо известны — это выбор папки для запуска устанавливаемого программного продукта. Выбирайте папку и нажимайте на кнопку «Install».

Начнется процесс установки, который займет несколько минут (Рис. 1.6). После завершения процесса установки среду разработки Android Studio можно запускать.

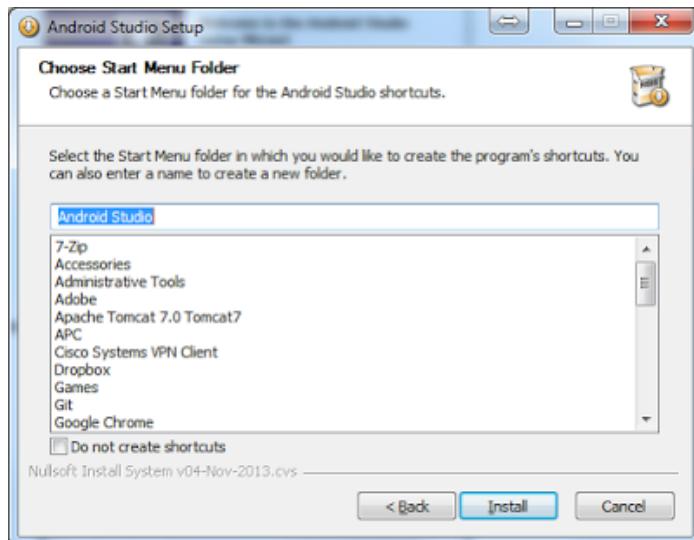


Рис. 1.5. Выбор Start Menu Folder

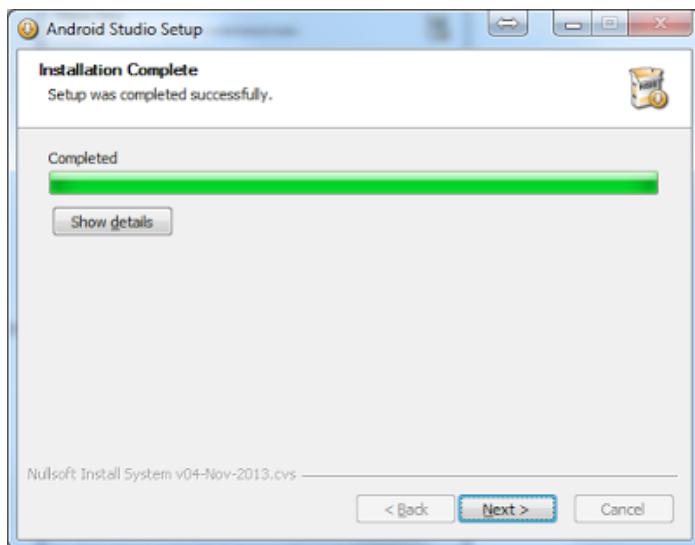


Рис. 1.6. Процесс установки

Однако, при первом запуске Android Studio перед созданием первого проекта необходимо установить необходимые версии SDK, иначе проект будет создан под существующую версию SDK (версия была установлена вместе с дистрибутивом Android Studio), которая может не отвечать вашим требованиям и после скачивания/установки нужной вам SDK проект придется пересоздавать.

Итак, запускаем Android Studio. Видим окно (Рис. 1.7).

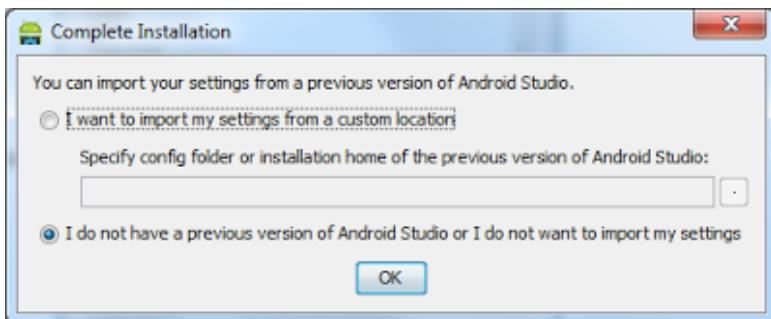


Рис. 1.7. Импорт возможных настроек от предыдущей версии Android Studio

В этом окне предлагается, по желанию, сделать импорт настроек предыдущей инсталляции Android Studio (если такая была). Предпочтительно отказаться от импорта старых настроек и настроить Android Studio чтобы познакомится с опциями настроек новой версии студии.

Нажав на кнопку «OK» появляется следующее окно (Рис. 1.8). Это окно первого запуска.

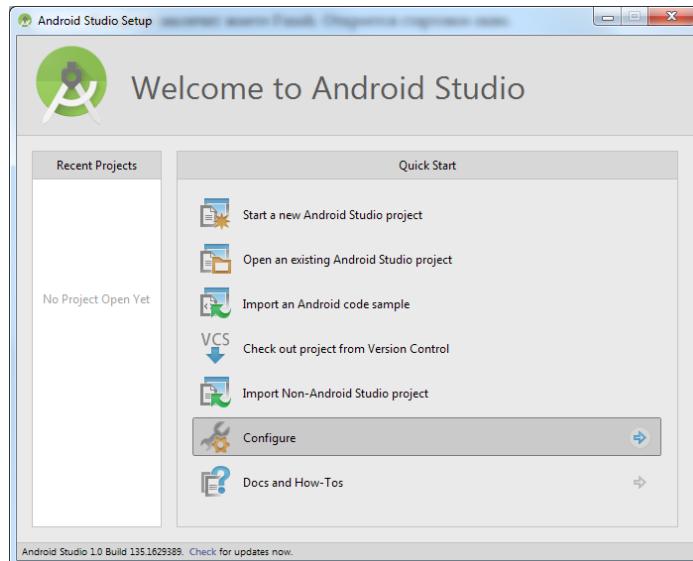


Рис. 1.8. Первый запуск Android Studio

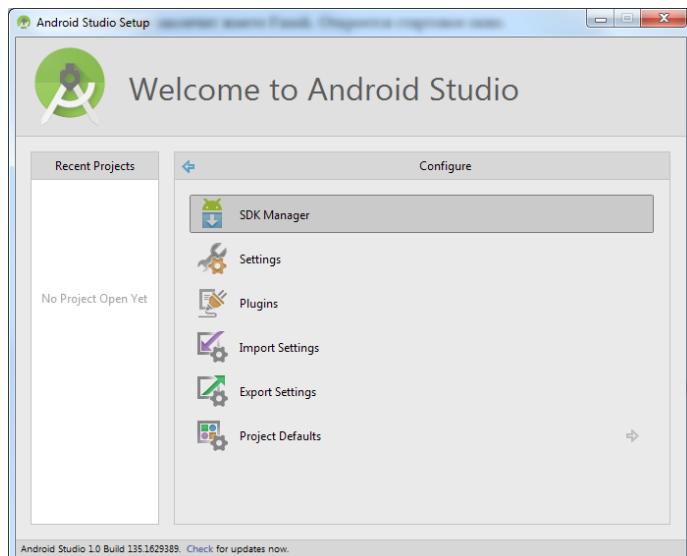


Рис. 1.9. Запуск утилиты SDK Manager

В этом окне можно выбрать создание нового проекта («New Project...»), но не торопитесь — давайте сначала установим необходимые версии SDK — ведь при создании нового проекта, указывается версия SDK (в виде выбора версии API). Для этого нужно выбрать опцию «Configure». (Рис. 1.9)

2. Понятие SDK, SDK-Manager, Эмулятор, AVD

Как вы наверняка уже знаете из предыдущих курсов, SDK это Software Developer Kit (*Инструмент разработчика программного обеспечения*) — набор библиотек классов, функций, утилит и всевозможных ресурсов для создания программных продуктов под определенную платформу. Разработчик непосредственно использует средства SDK чтобы создавать свои программы. При разработке Android приложений приходится учитывать тот факт, что постоянно выходят новые версии операционных систем Android, и вместе с ними расширяется функциональность операционных систем, который становится доступен для разработчиков через новые версии SDK. Номер SDK соответствует номеру API (*Application Program Interface*). При создании нового проекта нужно выбрать версию SDK/API (target SDK), для которой предназначается данное разрабатываемое приложение, а так же минимальную версию SDK/API, которая будет поддерживаться этим приложением. Так вот номера SDK и API одинаковы.

В таблице 2.1 приведены некоторые соответствия номеров (уровней) API и версий ОС Android.

Таблица 2.1 Версии Android и уровни API

Версия ОС Android	API level	Код версии ОС Android
Android 5.1	22	LOLLIPOP_MR1
Android 5.0	21	LOLLIPOP
Android 4.4W	20	KITKAT_WATCH
Android 4.4	19	KITKAT
Android 4.3	18	JELLY_BEAN_MR2
Android 4.2, 4.2.2	17	JELLY_BEAN_MR1
Android 4.1, 4.1.1	16	JELLY_BEAN
Android 4.0.3, 4.0.4	15	ICE_CREAM SANDWICH_MR1
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM SANDWICH

Другими словами, разработчик приложения, определяется с версией ОС Android, и устанавливает необходимый SDK для разработки. Установка/удаление/обновление версий SDK осуществляется при помощи утилиты «SDK Manager», которая входит в установленный пакет программ вместе с Android Studio.

2.1. SDK Manager

SDK Manager можно запустить или через опцию «Configure» при первом запуске Android Studio (см. Рис. 1.8) или непосредственно из Android Studio из «Панели Инструментов» (Рис. 2.1) или с помощью меню «Tools->Android->SDK Manager».

После запуска SDK Manager вы увидите следующее окно (Рис. 2.2). В окне представлен список древовидной структуры, в котором узлами являются элементы с версией операционной системы Android и соответствующие им версии API. Распахнув интересующий API можем наблюдать подчиненные элементы со статусами «Установлен/Не установлен». Это так называемые компоненты SDK.

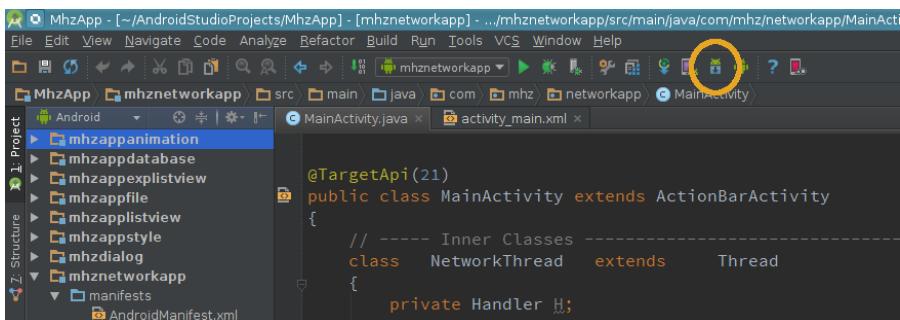


Рис. 2.1. Пиктограмма SDK Manager на панели инструментов Android Studio

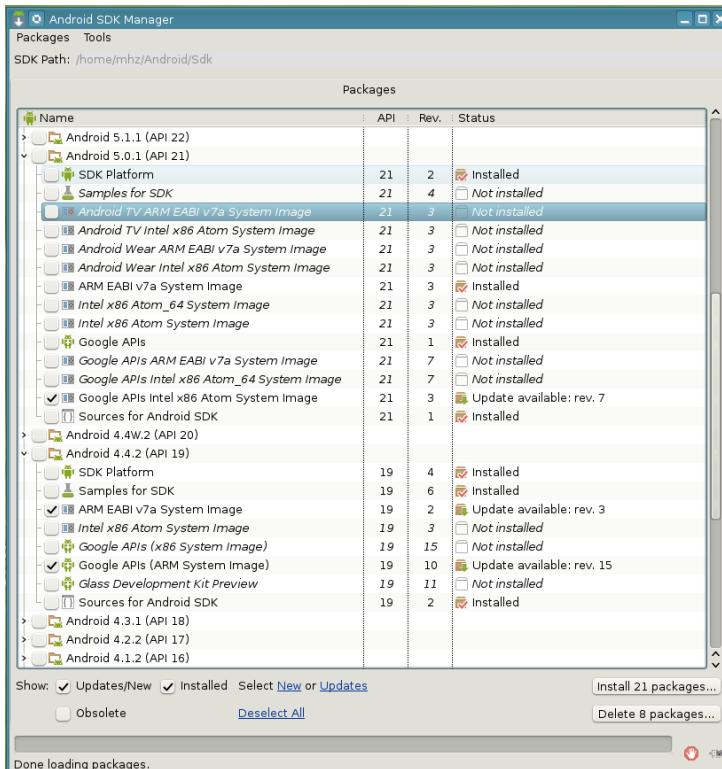


Рис. 2.2. Главное окно утилиты SDK Manager

Для установки необходимого API вовсе не нужно выбирать все эти компоненты. Установка всех компонентов занимает достаточно длительное время и место на жестком диске. Кроме версий API в списке есть еще два элемента — «Tools» и «Extras». В «Tools» находятся утилиты, необходимые для разработки в Android Studio. В «Extras» находятся дополнительные библиотеки и драйвера.

Давайте рассмотрим, какие компоненты входят в состав API, на примере API 21:

- **SDK Platform** — непосредственно сам SDK
- **Samples for SDK** — простейшие примеры по использованию различного функционала, представленного в SDK.
- **Android TV ARM EABI v7a System Image** — образ операционной системы Android для телевизоров. Он используется для эмулирования операционной системы в процессе отладки и тестирования приложения.
- **Android TV Intel x86 Atom System Image** — образ операционной системы для телевизоров. Этот образ используется для эмулирования ОС Android в режиме x86 (без эмуляции ARM). Требует специально установленного на компьютере HAXM программного обеспечения для ускорения работы эмулятора (Подробнее о HAXM: <https://software.intel.com/ru-ru/android/articles/intel-hardware-accelerated-execution-manager>).
- **Android Wear ARM EABI v7a System Image** — образ операционной системы Android для умных часов.
- **Android Wear Intel x86 Atom/Atom_64 System Image** — образ операционной системы для умных часов, рабо-

тающий в режиме x86 (без эмуляции ARM). Требует HAXM.

- **ARM EABI v7a System Image** — образ операционной системы для мобильных устройств. Используется для эмулирования работы ОС Android в процессе отладки и тестирования разрабатываемых приложений.
- **Intel x86 Atom/Atom_64 System Image** — образ ОС Android для мобильных устройств. Используется для эмулирования работы ОС Android (режим x86 без ARM) в процессе отладки и тестирования. Требуется HAXM
- **Google APIs** — компонент SDK для разработки приложений, взаимодействующих с сервисами Google, такими как Google Maps и т.д.
- **Google APIs ARM EABI v7a System Image** — образ ОС Android с сервисами Google для эмулятора.
- **Google APIs Intel x86 Atom/Atom_64 System Image** — образ ОС Android с сервисами Google для эмулятора (режим x86 без ARM). Требуется HAXM.

Эмулятор — это специальная программа, которая позволяет эмулировать работу устройства (например мобильного телефона) непосредственно на персональном компьютере.

Как видите, очень много компонентов представляют образы для эмуляторов, причем не все образы нужны для разработки приложений под мобильные устройства. Что касается образов Intel, то они должны обладать более высокой производительностью, но их использование требует чтобы на вашем компьютере был процессор Intel (или процессор поддерживающий виртуализацию Intel)

и установленный менеджер HAXM (*Hardware Accelerated Execution Manager* — приложение с поддержкой аппаратной виртуализации, которое использует технологию виртуализации Intel для ускорения эмуляции приложений Android на компьютере разработчика). В любом случае, работа эмуляторов является достаточно медленной, поэтому лучше всего установить оба типа образа (ARM EABI и Intel Atom) для эмулятора — чтобы поэкспериментировать и выбрать тот образ, с которым ваш компьютер будет работать быстрее.

Итак, для того чтобы успешно разрабатывать приложения под мобильные устройства Android необходимо установить следующий минимальный набор компонентов SDK:

- SDK Platform
- Samples for SDK
- ARM EABI v7a System Image
- Intel x86 Atom/Atom_64 System Image

Выберите необходимый вам API (проконсультируйтесь с преподавателем — на какую версию API делается акцент при изучении данного курса) и выполните установку, согласившись с условиями лицензионного соглашения (см. Рис. 2.3).

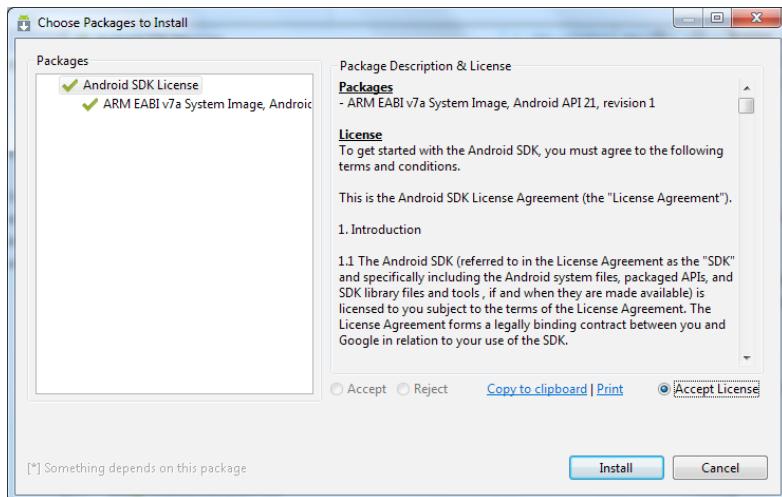


Рис. 2.3. Принятие условий перед установкой выбранных компонентов SDK

2.2. Android Virtual Device Manager (AVD Manager)

Необходимо познакомится с еще одной важной утилитой, которая входит в пакет программ вместе с Android Studio. Это Android Virtual Device Manager — менеджер виртуальных устройств. Виртуальное устройство (AVD — *Android Virtual Device*) — это конфигурация некоторого реального устройства для эмулирования на персональном компьютере. Перед тем как запускать эмулятор для тестирования приложения, необходимо чтобы было создано одно или несколько виртуальных устройств — именно виртуальное устройство эмулируется эмулятором.

Запустить AVD Manager можно с помощью меню «Tools» / «Android» / «AVD Manager» или с помощью клика по пиктограмме на панели инструментов (Рис. 2.4).

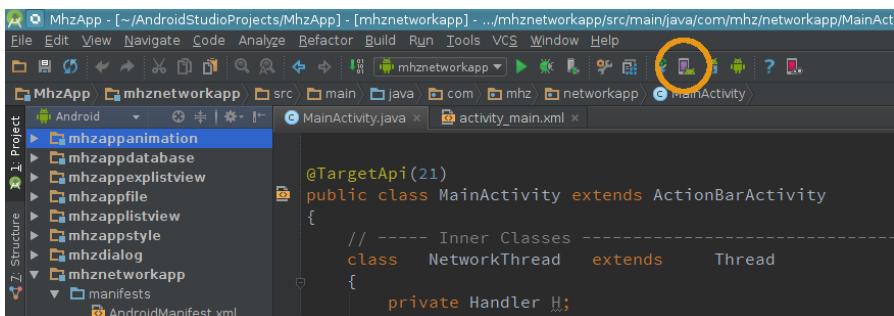


Рис. 2.4. Пиктограмма AVD Manager на панели инструментов Android Studio

2.3. Создание виртуальных устройств для эмуляторов

Давайте научимся создавать виртуальные устройства. Запустите AVD менеджер. Вы увидите главное окно утилиты (Рис. 2.5) в котором отображается список ранее созданных виртуальных устройств с краткой информацией (версия API, разрешение экрана устройства, занимаемое место на жестком диске и т.д.).

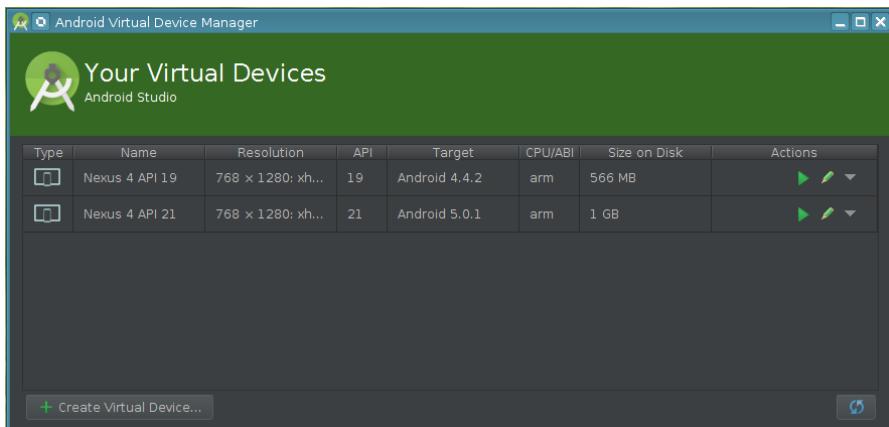


Рис. 2.5. Главное окно утилиты AVD Manager

Эти виртуальные устройства можно изменять и удалять. Но нас интересует создание — нажмите на кнопку «Create Virtual Device». Появится окно выбора типа устройства и основными доступными характеристиками — размер экрана, количество точек на дюйм (Рис. 2.6). Делайте свой выбор и жмите «Next».

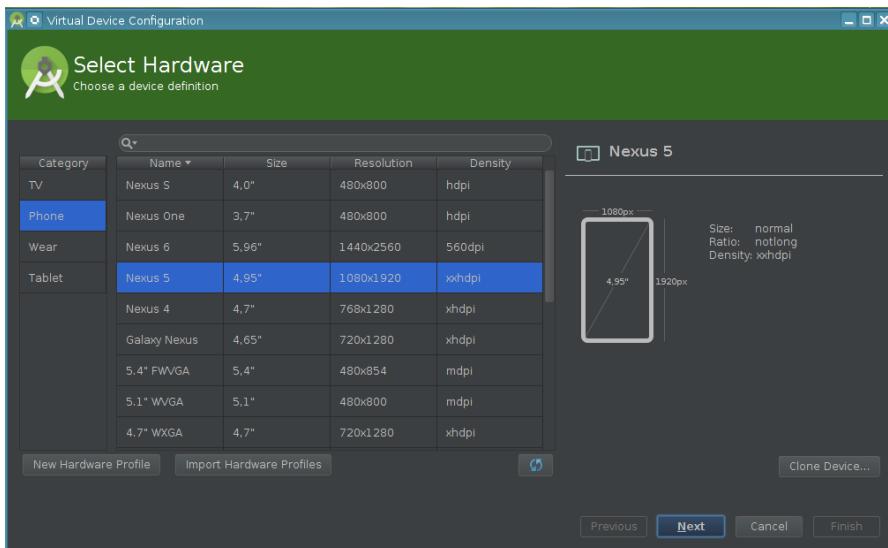


Рис. 2.6. Выбор конфигурации для виртуального устройства

В следующем окне вам нужно выбрать версию API и образ операционной системы для эмулятора (Рис. 2.7).

Обратите внимание, что Android Studio красным цветом рекомендует установить ПО HAXM даже для образа операционной системы, для которой HAXM не обязателен. Вы можете проигнорировать это предупреждение или выполнить установку HAXM — на данный момент это не принципиально, но в дальнейшем, когда вы будете заниматься разработкой приложений для ОС Android,

вам придется создать для себя наиболее оптимальные по быстродействию условия для разработки и HAXM вам в этом случае должен помочь.

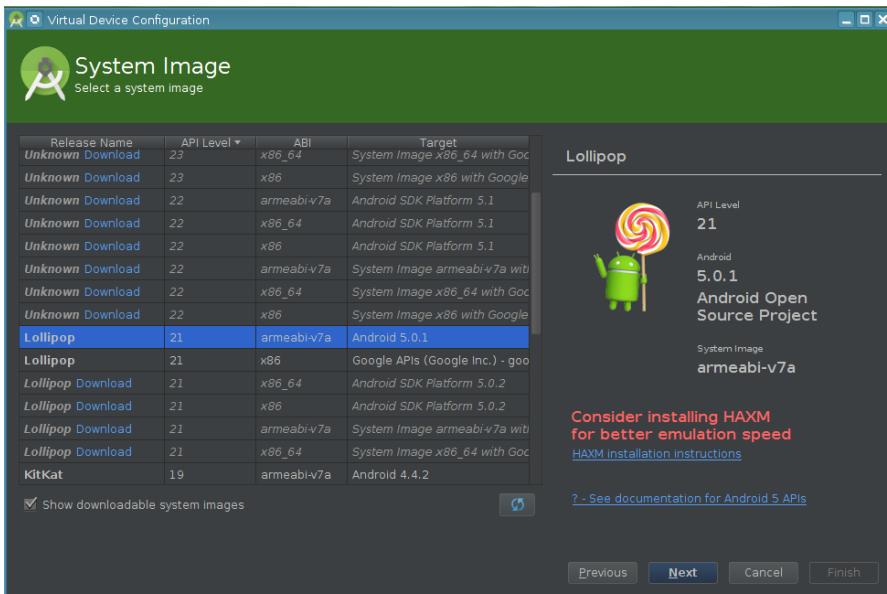


Рис. 2.7. Выбор версии API и образа ОС для эмулятора

Итак, выбрав API и образ, нажимаем «Next» и переходим к следующему окну (Рис. 2.8). В этом окне предлагается выполнить настройку параметров конфигурации эмулируемого устройства.

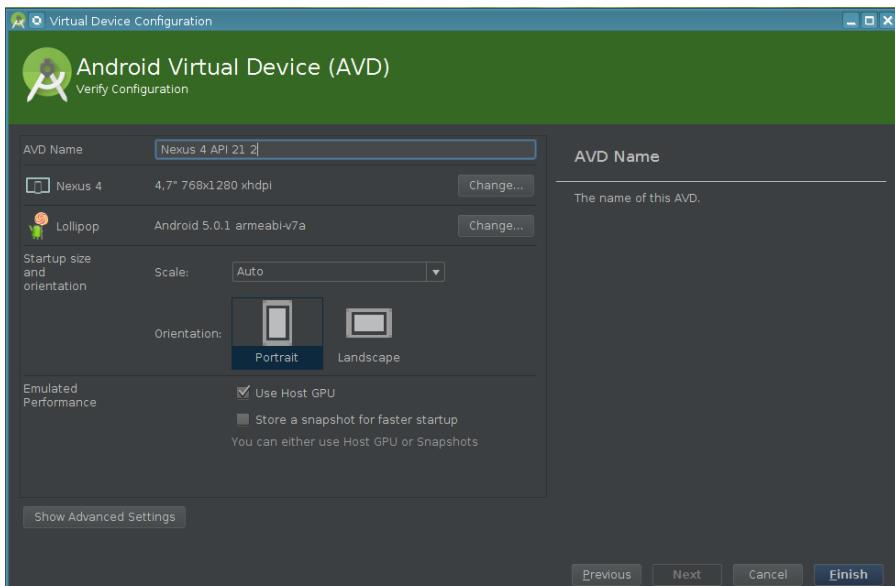


Рис. 2.8. Конфигурация эмулируемого устройства

Давайте рассмотрим опции этого окна более детально:

- Опция «**AVD Name**» — Понятное имя виртуального устройства. При запуске разрабатываемого вами приложения из Android Studio вам будет предлагаться список виртуальных устройств для выбора, на каком из устройстве необходимо запустить приложение. Список виртуальных устройств будет содержать эти самые понятные имена, которые вы будете назначать. Рекомендуется в имя включать модель устройства, версию API.
- «**Startup size and orientation / Scale**» — Масштаб экрана виртуального устройства для случая, если разрешение экрана виртуального устройства превышает размеры монитора компьютера, на котором разрабатывается приложение. По умолчанию опция установлена

в значение «Auto» — автоматический масштаб. Это хороший выбор для опции, но вы можете подобрать свой масштаб из списка.

- «**Startup size and orientation / Orientation**» — Ориентация устройства при запуске виртуального устройства в эмуляторе. Портретная и альбомная ориентация. Вне зависимости от выбора ориентации, есть возможность менять ориентацию устройства в процессе тестирования приложения в эмуляторе.
- «**Use Host GPU**» (*GPU* это *Graphics Processing Unit*) — Опция разрешает эмуляцию графики средствами видеокарты вашего персонального компьютера для увеличения скорости исполнения эмулируемого приложения в случае, если это приложение использует OpenGL технологию работы с графикой. Рекомендуется оставить эту опцию выбранной, так как ее отключение может повлечь к предупреждениям на этапе компиляции/исполнения вашего приложения, даже если ваше приложение не использует OpenGL.
- «**Store a snapshot for faster startup**» — Опция позволяет сделать снимок виртуального устройства и сохранить его на диск, чтобы в дальнейшем запуск виртуального устройства осуществлялся намного быстрее. Как уже упоминалось выше — работа виртуальных устройств в эмуляторе происходит достаточно медленно и данная опция должна обладать полезностью с точки зрения повышения скорости запуска виртуального устройства в эмуляторе. Однако, на практике было замечено, что в некоторых случаях использование снимка виртуального устройства приводило к необъяснимым

ошибкам в работе эмулируемого приложения, которые (ошибки) исчезали и более не появлялись если разработчики отказывались от использования снимка виртуального устройства. Объяснения, которые можно дать по этим случаям относятся к ситуации, что среда разработки Android Studio еще имеет целый ряд неточностей и недоработок, которые постепенно устраняются командной разработчиков Google.

Но это еще не все опции, которые предлагаются для конфигурирования созданного виртуального устройства. В левом нижнем углу диалогового окна есть кнопка «Show Advanced Settings» с помощью которой можно управлять дополнительными (но не менее важными) опциями виртуального устройства. Жмем на эту кнопку. Видим что в этом же окне появились следующие опции (Рис. 2.9):

- «**Camera / Front**» и «**Camera / Back**» — опции для эмуляции камеры (Фронтальная и задняя камеры соответственно). До тех пор, пока ваше приложение не работает ни с одной из этих камер изменять значения этих опций не нужно. Но когда придет время работать с камерой — обязательно включите эти опции.
- «**Network / Speed**» и «**Network Latency**» — опции относятся к эмуляции сетевых соединений. Понадобятся позже — когда будем изучать сетевое программирование для Android. На данном этапе значения этих опций не меняем.

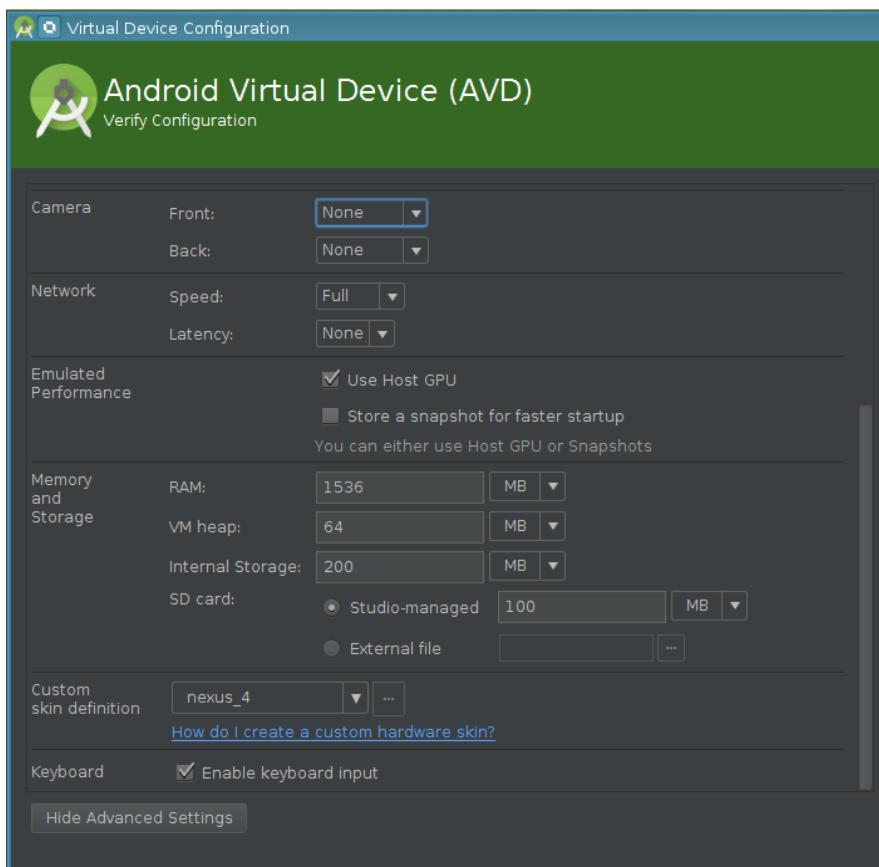


Рис. 2.9. Расширенные настройки виртуального устройства

- «**Memory and Storage / RAM**» — эта опция управляет размером динамической памяти которое будет выделено для виртуального устройства на компьютере где работает эмулятор. Соответственно, это память, которая может потребоваться вашему разрабатываемому приложению. Пока вы только обучаетесь, значение этой опции можно смело установить в значение

256 Мбайт — таким образом вы сэкономите размер вашего реального ОЗУ компьютера при тестировании вашего приложения (ведь ваши приложения на начальном этапе обучения будут не сложными). И соответственно, быстродействие эмулятора тоже возрастет из-за уменьшения количества обращений к файлу подкачки вашего компьютера (Чем больше свободной памяти ОЗУ — тем меньше взаимодействия с файлом подкачки, тем быстрее работает ваш компьютер. Все просто. Вы должны это знать).

- «**Memory and Storage / VM Heap**» — размер динамической памяти необходимый Java виртуальной машине для исполнения приложений в виртуальном устройстве. Значение этой опции не меняем.
- «**Memory and Storage / Internal Storage**» — размер эмулируемого внутреннего носителя. У реальных смартфонов есть два типа носителей — внешний и внутренний. Внешний накопитель — это карты памяти типа «microSD», которые владелец смартфона может менять по своему усмотрению. Внутренний накопитель — это не меняемая память устройства. Значение этой опции — на ваш выбор. Рекомендуемое значение которое предлагается — должно вас вполне устроить. Величина этой опции влияет на размер образа виртуального устройства на жестком диске вашего компьютера.
- «**Memory and Storage / SD Card**» — управляет размером и типом эмулируемого внешнего носителя. Можно привязать к внешнему файлу («External file», необходимо будет выбрать файл на диске вашего компью-

тера, который будет выступать в качестве внешнего носителя) или к виртуальному устройству, которое вы конфигурируете («Studio Managed»).

- «Custom skin definition» — возможность выбора другого «скина» для внешнего вида виртуального устройства.

Подводя итог, еще раз делается акцент на том что, в расширенных настройках виртуального устройства можно уменьшить размер RAM-памяти для ваших приложений (на первых этапах обучения вам вряд ли понадобится большой размер RAM, а быстродействие работы эмулируемого виртуального устройства будет выше).

Далее, нажимайте на кнопку «Next» и созданное вами виртуальное устройство появится в списке виртуальных устройств (Рис. 2.5) AVD менеджера. К этим виртуальным устройствам мы вернемся позже, когда будем запускать приложения.

3. Создание проекта Android Studio

Чтобы создать проект, необходимо выбрать пункт в меню «File» / «New Project» или выбрать пункт «Start a new Android Studio Project» в окне, которое вы можете увидеть при первом запуске Android Studio (Рис. 1.8). Появится диалоговое окно создания нового проекта (Рис. 3.1).

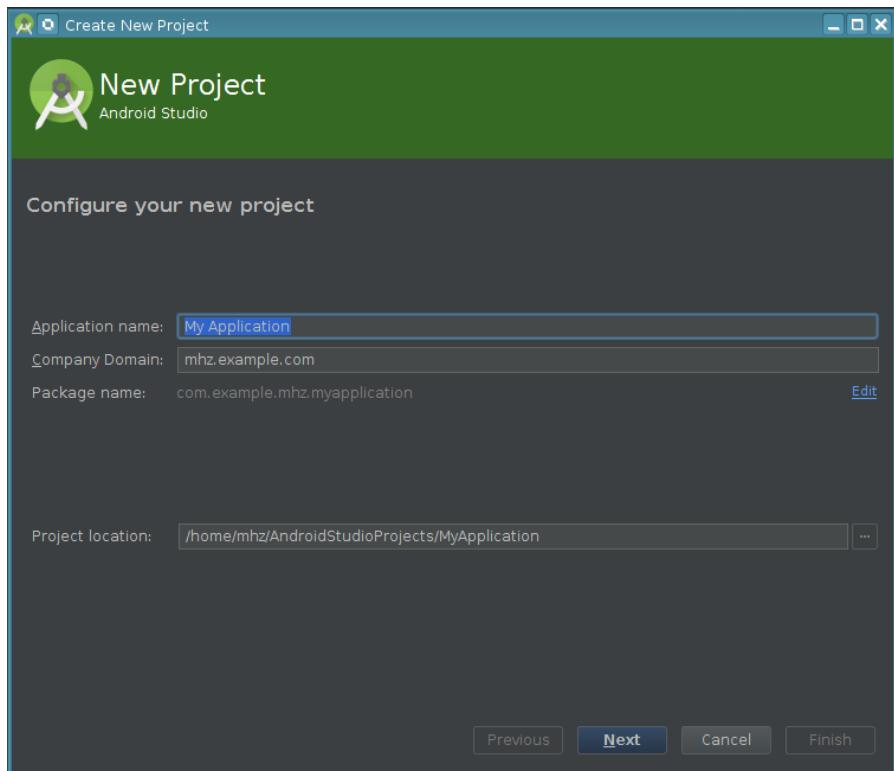


Рис. 3.1. Окно создания нового проекта в Android Studio

В окне предлагается указать следующие настройки:

- «**Application Name**» — Имя создаваемого проекта (приложения). На ваш выбор.
- «**Company Domain**» — домен вашей организации, если имеется. В случае отсутствия таковых, можно указать любой понравившийся вам. Значение домена будет использовано Android Studio в названии пакетов ваших Java-классов (Что такое пакет «package» вам должно быть известно из курса «Программирование на языке Java»).
- «**Package Name**» — Имя пакета для ваших Java-классов (см. Предыдущий пункт). При желании, значение этой опции можно изменить.
- «**Project location**» — расположение на жестком диске файлов создаваемого проекта.

Для первого создания проекта можете значения этих опций оставить по умолчанию. Нажимайте на кнопку «Next». В появившемся окне (Рис. 3.2) необходимо выбрать устройство, под которое создается проект и минимальную версию SDK. Минимальная SDK — это версия SDK, ниже которой ваше приложение работать не сможет, но сможет работать на более поздних версиях SDK. Наш курс посвящен разработке приложений для мобильных устройств, поэтому выбираем опцию «Phone and Tablet». Внимательно прочитайте рекомендацию о выборе минимальной версии SDK, которую предлагает Android Studio. На Рис. 3.2 предлагаемая минимальная версия SDK (и соответственно все версии выше) охватывает более 90% устройств. Очень даже хороший показатель.

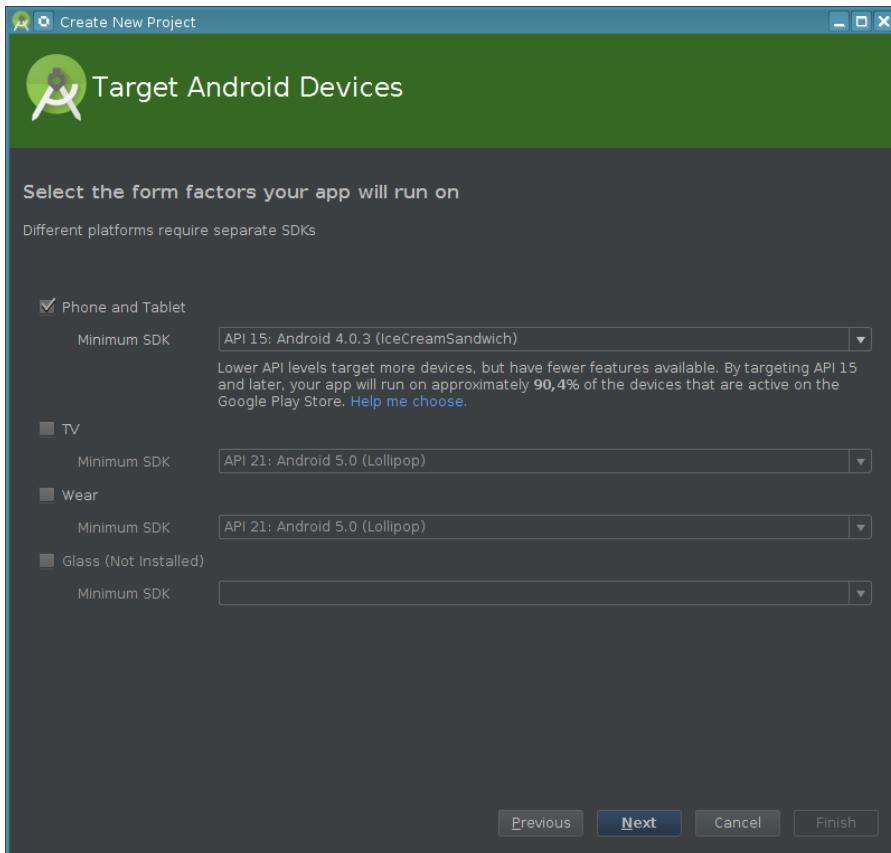


Рис. 3.2. Выбор минимального SDK для создаваемого проекта

Стоит отметить, что не нужно гнаться за 100% устройств, на которых сможет функционировать ваша приложение. Это просто невозможно! Хотя бы потому, что зачастую, более поздние версии Android SDK отменяют функционал более ранних версий. То есть ранние и поздние версии SDK становятся взаимоисключающими. И чтобы приложение могло работать на таких взаимоисключающих версиях, разработчику необходимо прикладывать очень

большие усилия. Поверьте — эти усилия экономически не оправдывают себя.

Нажимаем кнопку «Next» появляется окно (Рис. 3.3) выбора каркаса для макета Активности («Activity» — *Активность, Деятельность* — это понятие которое можно сравнить с главным окном приложения, но это не совсем одно и тоже — «Активность» и «Главное окно приложения»).

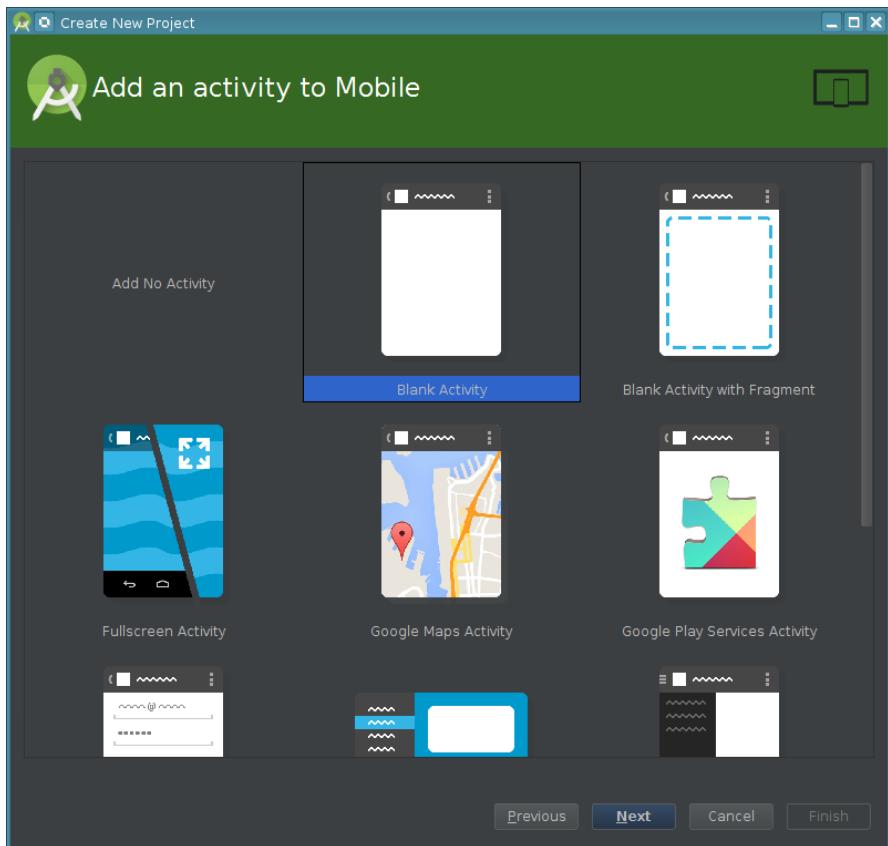


Рис. 3.3. Выбор Активности для проекта

В программировании для Android не говорят «Главное окно приложения», а говорят «Активность». Что это такое — будет рассказано позже). Каркас макета для Активности это каркас внешнего вида для размещения «Элементов Управления» (Опять же, термин «Элемент Управления» для Android-программирования не используется. Используется термин «Виджет» — «Widget»).

Окно предлагает выбор Активностей для разного типа приложений. Каждому макету из списка будет соответствовать свой класс Активности, который приспособлен под требуемый функционал приложения (Посмотрите на названия Активностей — их названия говорят сами за себя: «Fullscreen Activity» — *Макет для создания полноэкранных приложений*, «Google Maps Activity» — *приложение, использующее Google-карты и т.д.*). В большинстве случаев нашего курса нам нужен «Blank Activity». Выбирайте «Blank Activity» и нажимайте «Next».

В следующем окне (Рис. 3.4) предлагается указать:

- «**Activity Name**» — имя для класса Активности. По умолчанию — «MainActivity».
- «**Layout Name**» — имя макета для активности, оно же и имя файла xml где хранится разметка макета. По умолчанию — «activity_main»
- «**Title**» — имя Активности. Для главной Активности приложения это будет имя приложения. По умолчанию — «MainActivity».
- «**Menu Resource Name**» — имя ресурса (и соответственно имя файла xml) меню приложения.

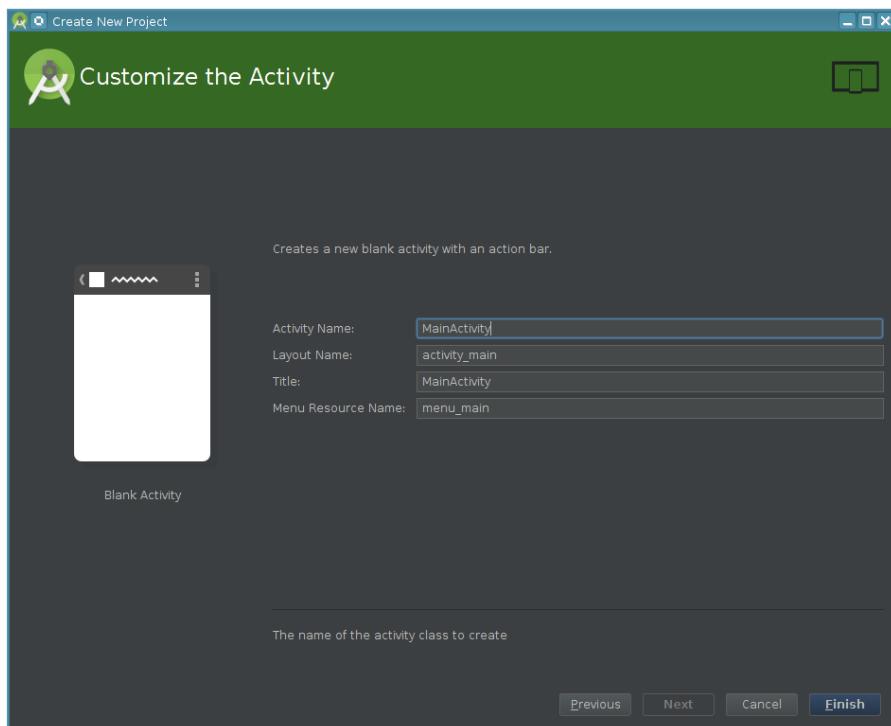


Рис. 3.4. Ввод названия класса Активности, имени раскладки макета и т. д.

Для первого проекта (и возможно для многих других ваших проектов) можно оставить значения по умолчанию.

Нажмите кнопку «Finish». Android Studio начнет создавать новый проект. Это займет некоторое время. Прогресс создания будет отображаться в окне (Рис. 3.5).

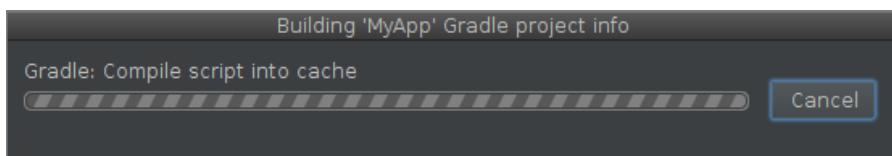


Рис. 3.5. Прогресс создания проекта

Обратите внимание (Рис. 3.5) что процессом создания проекта занимается некий «Gradle». Так вот, знакомьтесь — «Gradle» это система автоматической сборки проектов.

Эта система имеет свой собственный язык и используется Android Studio для компиляции исходного кода файлов в бинарный (промежуточный) код, сборки промежуточного кода, сборки ресурсов приложения и т.д. Официальный сайт «Gradle» — <http://gradle.org>. Система Gradle является достаточно мощным средством сборки проектов, однако потребляет достаточное количество ресурсов и процессорного времени, а так же нуждается в доступе к Интернет. Все что нужно начинающему разработчику Android так это предоставить Gradle выполнять свою работу ☺ Другими словами — Android Studio с помощью Gradle берет на себя все заботы о проекте, и разработчику, как правило, не приходится вмешиваться в этот процесс.

Когда проект будет создан, вы увидите то, что изображено на Рис. 3.6. — окно редактора формирования/изменения внешнего вида (макета) Активности визуальными средствами, по возможностям очень сильно напоминающее «Конструктор форм» возможно хорошо знакомый вам из среды разработки Microsoft Visual Studio курса Windows Forms.

В этом редакторе разработчик с помощью мыши перетаскивает на макет Активности необходимые виджеты (Элементы Управления) и с помощью вкладки «Properties» модифицирует свойства этих виджетов. В общем, все очень похоже на MS Visual Studio.

3. Создание проекта Android Studio

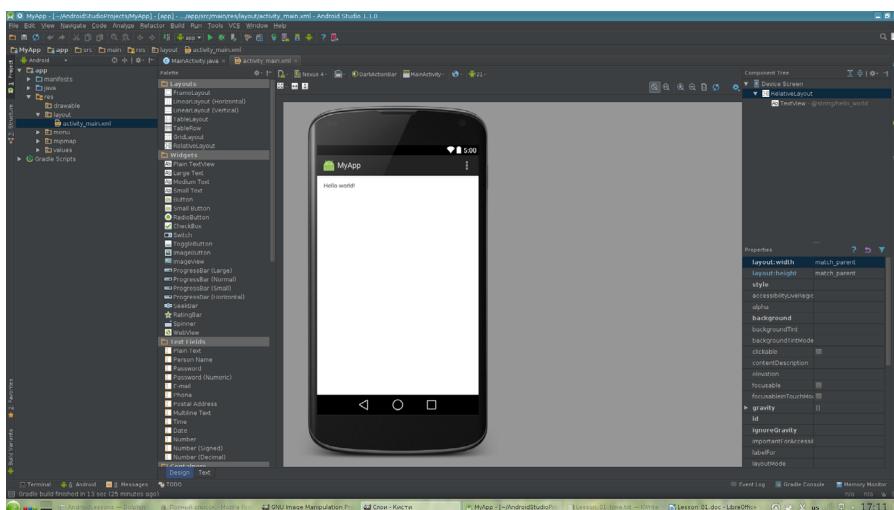


Рис. 3.6. Редактор формирования/изменения внешнего вида (макета) Активности

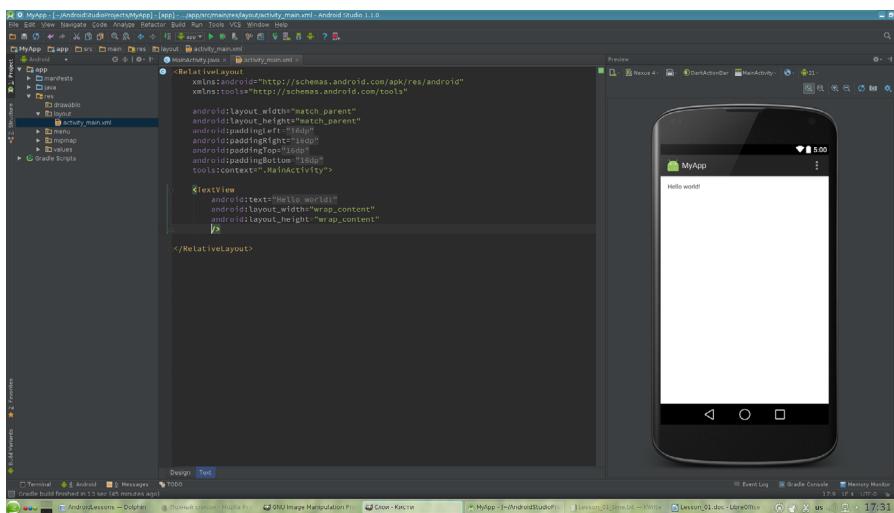


Рис. 3.7. Окно редактирования макета Активности с помощью XML

Учитывая ваш опыт работы с Microsoft Visual Studio, принципы работы в таком редакторе расписываться не будут по причине интуитивно понятного интерфейса редактора. Акцент, при изложении учебного материала делается на формирование макета Активности средствами XML. Перейти в режим редактирования макета Активности можно, нажав на вкладку «Text» (найти ее можно в левой нижней части Рис. 3.6, рядом с подсвеченной синим цветом вкладкой «Design»). Давайте выберем вкладку «Text» и перейдем в окно редактирования макета Активности средствами XML (Рис. 3.7). Видим XML документ, с очень красивыми, но пока неизвестными тегами и модель телефона, на которой будут сразу отображаться все вносимые нами изменения. На этом моменте мы временно остановимся, чтобы познакомиться со структурой созданного проекта. После чего, в следующих главах, мы продолжим создавать наше первое Android приложение.

4. Структура проекта разрабатываемого приложения в Android Studio

4.1. Модули проекта

На самом деле, проект Android Studio — это набор модулей. При создании проекта создается проект и один модуль. Один модуль представляет одно Android приложение. Далее в проект можно добавлять новые модули. И на основе каждого модуля можно получать Android приложение. В отличии от других IDE, в Android Studio может быть открыт только один проект. Открытие/создание нового проекта приводит к закрытию предыдущего проекта и открытию/созданию другого. Но зато в каждом проекте можно создавать много модулей (Рис. 4.1).



Рис. 4.1. Модульная структура проекта Android Studio

Для того чтобы добавить модуль в проект, необходимо выбрать пункт меню «File» / «New Module» и повторить шаги, указанные на Рисунках 3.1, 3.2, 3.3, 3.4.

4.2. Структура модуля проекта

Рассмотрим подробнее файлы, входящие в каждый модуль (Рис. 4.2). Структура фолдеров (папок) модуля совпадает со структурой каталогов модуля на жестком диске.

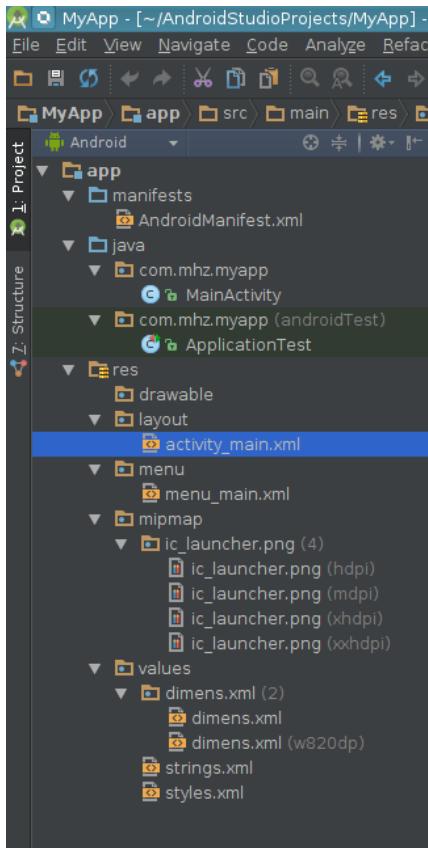


Рис. 4.2. Структура модуля проекта Android Studio

Итак, содержимое модуля, которое мы увидим при создании нового модуля:

- **app** — название модуля. Главная папка модуля.
- **app/manifests** — в этой папке находится файл манифеста приложения.
- **app/manifests/AndroidManifest.xml** — манифест приложения.
- **app/java** — папка, содержащая файлы исходных кодов приложения.
- **app/java/com.mhz.myapp/MainActivity.java** — исходный код класса **MainActivity** — активности приложения, где **com.mhz.myapp** — имя пакета, которое было указано при создании модуля (Рис. 3.1. Опция «Package Name»).
- **app/java/com.mhz.myapp/ApplicationTest.java** — файл исходного кода класса с функционалом для тестирования приложения (разумеется, этот функционал необходимо будет тоже написать).
- **app/res** — папка с ресурсами приложения.
- **app/res/drawable** — папка, в которую помещаются файлы, использующиеся для отображения графического контента. Например файлы изображений в формате .png или .jpg
- **app/res/layout** — папка, содержащая макеты внешнего вида Активностей.
- **app/res/layout/activity_main.xml** — макет внешнего вида Активности. Имя этого файла указывалось в окне на Рис. 3.4.

- **app/res/menu** — папка с ресурсами для меню приложения.
- **app/res/menu/menu_main.xml** — файл с пунктами главного меню приложения. Имя файла указывалось в окне на Рис. 3.4.
- **app/res/mipmap** — специальная папка ресурсов, предназначенная для хранения иконок приложения. На Рис. 4.2 можно увидеть, что иконок приложения несколько — на каждое разрешение экрана устройства.
- **app/res/values** — папка, для хранения ресурсов приложения, которые содержат значения (т.е. некоторые величины). Например, такие значения как величины отступов, константы используемых в приложении RGB-цветов, строковые константы и т.д.
- **app/res/dimens.xml** — файл ресурсов в формате xml, содержащий константные значения для величин всех возможных размеров (например размер отступов контейнера, содержащего виджеты, от границ Активности). В файлах исходных кодов приложения для обращения к этим ресурсам используют специальные идентификаторы, которые назначаются автоматически Android Studio.
- **app/res/strings.xml** — файл ресурсов, содержащий строковые константы. В Android проектах настоятельно рекомендуется все строковые константы определять в ресурсах приложения, чтобы облегчить процесс дальнейшей локализации приложения под разные языки. В файлах исходных кодов приложения для обращения к строке используется идентификатор, который

автоматически присваивается при создании строки в ресурсах. Об этом будет упоминаться ниже.

- `app/res/styles.xml` — файл ресурсов, в котором определяются всевозможные стили для виджетов приложения. О стилях будет рассказано в одном из последующих уроков.

4.3. Манифест приложения

Манифест (`app/manifests/AndroidManifest.xml`) это своего рода конфигурационный файл приложения. Но это конфигурационный файл не для самого приложения, а для операционной системы, запускающей приложение. Т.е. манифест — это обязательная для операционной системы информация о требуемом для приложения окружении, необходимыми привилегиями, необходимом списке разрешений, необходимыми доступами к ресурсам устройства и т.д. Манифест приложения еще содержит описание компонентов приложения. Так же в манифесте содержится имя приложения, ссылка на иконку приложения из ресурсов приложения, ссылка на главную тему Активности и прочее.

Листинг 4.1. Пример простого манифеста приложения

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    package="com.mhz.myapp">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
```

```
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name=".MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name=
                "android.intent.action.MAIN" />
            <category android:name=
                "android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
</manifest>
```

Когда вы перейдете к написанию более сложного проекта, то вам часто придется обращаться к манифесту для того чтобы выполнить конфигурацию какого-то из необходимых вам компонентов. По мере изучения материала курса, мы будем возвращаться к файлу манифеста приложения.

Структура файла манифеста приложения подробно описана на сайте разработчиков Android. Ознакомится с ней можно по ссылке: <http://developer.android.com/guide/topics/manifest/manifest-intro.html>.

Так же есть переведенная информация, например: <http://devcolibri.com/3010>.

4.4. Файл исходного кода класса Активности

При создании нового модуля, файл с классом Активности будет выглядеть, как показано в Листинге 4.2. Сам файл в структуре проекта находится по такому пути: app/java/имя_пакета/MainActivity.java.

Листинг 4.2. Первоначальный файл исходного кода класса Активности

```
package com.mhz.myapp;

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
        int id = item.getItemId();
        if (id == R.id.action_settings)
        {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

В классе Активности нашего приложения уже реализованы несколько методов:

- **onCreate** — обработчик события создания Активности. Подробнее об этом и других событиях Активности

будет рассказано ниже в этом уроке. В этом методе две команды. Это вызов одноименного метода родительского класса `super.onCreate(savedInstanceState)` и `setContentView(R.layout.activity_main)`. Метод `setContentView` назначает активности виджет, макет которого находится в файле `activity_main.xml`. Обратите внимание, что xml-файл с макетом для Активности передается в виде константы класса R: `R.layout.activity_main`. Этот класс создается автоматически и содержит идентификаторы всех ресурсов нашего приложения. Вы можете найти файл R.java в каталоге вашего проекта: Путь_к_каталогу_проектов/MyApp/app/build/generated/source/r/debug/com/mhz/myapp/.

- **onCreateOptionsMenu** — обработчик события создания опций главного меню. В этом обработчике можно управлять например созданием дополнительных пунктов главного меню. Принимаемый параметр `menu` — ссылка на объект меню Активности, для которого будут создаваться пункты меню. Внутри метода одна команда — `getMenuInflater().inflate(R.menu.menu_main, menu)` — создание пунктов меню из ресурсов. Опять же, обратите внимание на константу `R.menu.menu_main` — она задает идентификатор файла ресурсов app/res/menu/menu_main.xml который содержит описание пунктов меню.
- **onOptionsItemSelected** — обработчик события выбора какого-то пункта главного меню. Метод принимает один параметр `item` — ссылка на объект `MenuItem` выбранного пункта меню. Сам метод определяет

по идентификатору объекта **MenuItem** какой пункт меню выбран чтобы выполнить соответствующее действие.

По мере написания приложения, в класс Активности добавляются другие обработчики событий и прочие вспомогательные методы и вложенные классы.

4.5. Файл исходного кода разметки внешнего вида Активности

Давайте посмотрим на xml файл с разметкой макета Активности (Листинг 4.3), который был создан автоматически при создании нового модуля. Сам файл в структуре проекта находится в app/res/layout/activity_main.xml.

Листинг 4.3. XML разметка внешнего вида Активности

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

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

Корневым элементом этого XML документа является элемент `<RelativeLayout>`, это контейнер, который предназначен для размещения или других контейнеров или виджетов. Этот контейнер является еще и менеджером раскладки (*Layout Manager*). Менеджеры раскладки (или менеджеры компоновки) вам уже встречались в курсах WPF и Java. Менеджер раскладки располагает свои дочерние компоненты по определенным правилам, лишая разработчика головной боли по написанию самостоятельного функционала для размещения компонентов (виджетов, элементов управления). Неплохо, не правда ли? ☺ Так вот, менеджер раскладки `RelativeLayout` располагает свои дочерние виджеты относительно друг друга. На самом деле это не самый удобный менеджер раскладки. Возможно даже самый неудобный. Но это дело вкуса и предпочтений каждого разработчика. Мы будем с вами знакомится с различными менеджерами компоновки в этом и последующих уроках.

Итак, корневой элемент xml документа макета Активности может быть разным. Но главное, чтобы этот элемент подключал две xml-схемы и определял пространства имен: `xmlns:android="http://schemas.android.com/apk/res/android"` и `xmlns:tools="http://schemas.android.com/tools"`. В схеме, для которой назначается пространство имен android определены элементы и атрибуты для xml верстки макетов Активностей и других макетов. Так же, корневой xml-элемент (он же главный контейнер Активности) уже использует несколько атрибутов из этого пространства имен. Некоторые из этих атрибутов нам придется использовать настолько часто, что они вам должны будут сниться. ☺ Изучим их подробнее:

- **android:layout_width** — определяет по какому правилу будет задаваться ширина виджета или контейнера. Значение «**match_parent**» означает «растянуться на всю ширину родительского контейнера», а значение «**wrap_content**» означает «занять по ширине область, равную ширине своего содержимого». Так же в качестве значения для этого атрибута можно использовать число, которое будет означать фиксированный размер например в пикселях или пунктах. Разумеется, фиксированный размер хуже, чем адаптивный размер, так как размеры экранов для разных устройств разные и так же разные разрешения этих экранов.
- **android:layout_height** — определяет, по какому правилу будет задаваться высота виджета или контейнера. Значение «**match_parent**» означает «растянуться на всю высоту родительского контейнера», а значение «**wrap-content**» означает «занять по высоте область, равную высоте своего содержимого». Так же этому атрибуту можно задавать размер в виде числового значения. См. пункт выше.
- **android:paddingLeft** — внутренний отступ содержимого контейнера от границ контейнера. Обратите внимание, в каком виде задается значение для этого атрибута: «**@dimen/activity_horizontal_margin**» — в таком виде задаются константные значения размеров из ресурсов (об этом упоминалось в разделе этого урока «Структура модуля»). «**@dimen**» означает, что значение находится в файле ресурсов `app/res/values/dimens.xml`, а «**activity_horizontal_margin**» это идентификатор значения в ресурсе. Давайте заглянем в файл `app/res/`

values/dimens.xml и посмотрим, каким образом там объявляются подобные константы (см. Листинг 4.4).

Листинг 4.4. Файл ресурсов app/res/values/dimens.xml

```
<resources>
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
</resources>
```

Как видим, константе «activity_horizontal_margin» соответствует значение 16dp. Сuffix «dp» означает «Density-independent Pixels» — абстрактная единица, привязанная к физической плотности экрана. Эта единица соответствует одному пикслю при плотности экрана в 160 точек на дюйм (более подробно о единицах измерения размеров см. <http://developer.android.com/guide/topics/resources/more-resources.html#Dimension>). Эта единица измерения позволяет приложениям выглядеть одинаково на разных экранах и разрешениях.

- **android:paddingRight** — внутренний отступ справа (см. описание android:paddingLeft).
- **android:paddingTop** — внутренний отступ сверху (см. описание android:paddingLeft).
- **android:paddingBottom** — внутренний отступ снизу (см. описание android:paddingLeft).

Далее, внутри контейнера **RelativeLayout** на Листинге 4.3 находится элемент **TextView**. Это виджет для отображения текста. Атрибуты **TextView** «**android:layout_width**» и «**android:layout_height**» нам уже знакомы — здесь они играют точно такую же роль, как и для контейнеров. И еще

один атрибут «`android:text`» — содержит текст, который отображается в виджете `TextView`. Обратите внимание, что текст так же задается (и это настоятельно рекомендуется разработчикам android приложений) в виде константного значения из ресурсов: `android:text="@string/hello_world"` где «`@string`» означает, что ресурс находится в файле строковых ресурсов `app/res/values/strings.xml`, а «`hello_world`» это идентификатор ресурса. Давайте посмотрим в файл `app/res/values/strings.xml` (Листинг 4.5).

Листинг 4.5. Файл ресурсов `app/res/values/strings.xml`

```
<resources>
    <string name="app_name">MyApp</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
</resources>
```

Содержимое этого файла очень понятно. В этот файл мы позже добавим несколько строковых констант для нашего первого приложения.

5. Написание первого приложения «Hello World»

Наконец-то настало время написать наше первое Android приложение! По сложившейся в программировании традиции, это должно быть приложение, выводящее приветствие «Hello World!». Фраза «Hello World!» является не просто приветствием, это волшебная фраза программиста, которую нужно обязательно вывести/произнести в первом приложении любого языка программирования или платформы, которые вы изучаете, чтобы обучение было успешным! ☺

Как вы уже успели заметить, строка «Hello World» уже выведена в виджете **TextView** (да-да, разработчики Android Studio так же чтят традиции). Куда же будем выводить свое приветствие мы в нашем первом приложении? Конечно же в Тост! Тост (*Toast*) это всплывающие сообщения. Чтобы вывести всплывающее сообщение, необходимо воспользоваться классом **android.widget.Toast** (<http://developer.android.com/reference/android/widget/Toast.html>) и его статическим методом **makeText**:

```
public static Toast.makeText(Context context,  
    CharSequence text, int duration);
```

Метод формирует объект **Toast** с текстовым сообщением, передаваемым в параметре **text** и длительностью показа сообщения, передаваемой в параметре **duration**

(можно задавать только две разновидности длительности: `Toast.LENGTH_SHORT` и `Toast.LENGTH_LONG`). Параметр `context` это ссылка или на объект приложения `Application` (<http://developer.android.com/reference/android/app/Application.html>) или ссылка на объект Активности `Activity` (<http://developer.android.com/reference/android/app/Activity.html>). Оба этих класса являются производными от класса `android.content.Context`. Объект `android.content.Context` содержит глобальную информацию о среде окружения Android-приложения, а также позволяет получать доступ к ресурсам самого приложения. Ссылки на объект `android.content.Context` будет необходимо передавать в очень многие методы объектов разных классов. Ни один виджет не обходится без ссылки на `android.content.Context`.

Еще один метод класса `Toast`:

```
public void show();
```

Этот метод показывает сообщение, содержащееся в объекте `Toast`. Так же в классе `Toast` существует еще несколько методов, позволяющих управлять расположением и внешним видом всплывающего сообщения. С этими методами вы можете познакомиться в справочной информации на сайте для разработчиков Android (ссылка на API документацию класса давалась чуть выше).

В большинстве случаев, для отображения всплывающего сообщения `Toast` пишут код (Листинг 5.1):

Листинг 5.1. Пример использования `Toast` в одном из методов класса Активности

```
Toast.makeText(this, "Hello World!",  
    Toast.LENGTH_SHORT).show();
```

Итак, как выводить короткие всплывающие сообщения мы разобрались. Этап второй — где или когда мы их будем выводить. В этом уроке уже упоминалось, что у каждого приложения есть главное меню. В нашем примере мы добавим два пункта меню и по событию выбора каждого из этих пунктов меню будем выводить короткое сообщение.

Чтобы добавить два пункта меню, сделаем следующие шаги.

ШАГ ПЕРВЫЙ: добавим строковые константы в файл ресурсов app/res/values/strings.xml (Листинг 5.2).

Листинг 5.2. Добавление строковых констант в файл ресурсов app/res/values/strings.xml

```
<resources>
    <string name="app_name">MyApp</string>
    <string name="hello_world">Hello world!</string>
    <string name="android_forever">Android Forever!
                                </string>
    <string name="action_settings">Settings</string>
    <string name="action_hello_world">Show Hello World
                                </string>
    <string name="action_android_forever">
        Show Android Forever</string>
</resources>
```

Как видно из листинга, были добавлены две строковые константы с идентификаторами «**action_hello_world**» и «**action_android_forever**» (им соответствуют значения «Show Hello World» и «Show Android forever» соответственно). Префикс «**action_**» это всего лишь

правило хорошего тона при задании идентификаторов, в данном случае «**action_**» означает, что задаются строковые константы для пунктов меню. И была добавлена еще одна строковая константа «**android_forever**» со значением «Android Forever!». Как видите, в название идентификаторов вкладывается информация о строке, которая им соответствует — это облегчает процесс программирования.

ШАГ ВТОРОЙ: добавим описание пунктов меню в файл ресурсов app/res/menu/menu_main.xml (Листинг 5.3).

Листинг 5.3. Файл ресурсов с пунктами меню
app/res/menu/menu_main.xml

```
<menu
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:app=
        "http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MainActivity">

    <item
        android:id="@+id/action_settings"
        android:title="@string/action_settings"
        android:orderInCategory="100"
        app:showAsAction="never"
    />
    <item
        android:id="@+id/action_hello_world"
        android:title="@string/action_hello_world"
        android:orderInCategory="110"
        app:showAsAction="never"
    />
```

```
<item
    android:id="@+id/action_android_forever"
    android:title="@string/action_android_forever"
    android:orderInCategory="120"
    app:showAsAction="never"
/>
</menu>
```

В этом файле уже был описан один пункт меню:

```
<item
    android:id="@+id/action_settings"
    android:title="@string/action_settings"
    android:orderInCategory="100"
    app:showAsAction="never"
/>
</menu>
```

Наши пункты меню мы добавляем «по образу и подобию» данного пункта меню. Давайте рассмотрим, какие атрибуты используются элементом **item** для описания пункта меню:

- **android:id** — задает идентификатор пункта меню. Этот идентификатор затем будет использоваться для распознавания пункта меню, которое было выбрано (см. Листинг 4.2). Значение этого атрибута это и есть идентификатор. Обратите внимание на синтаксис «@+id/значение_идентификатора» — именно в таком виде и назначаются идентификаторы как для пунктов меню, так и для других элементов, описывающихся в разных xml файлах. Наши идентификаторы для создаваемых пунктов меню зададим аналогичным образом (см. Листинг 4.8).

- **android:title** — задает текст в пункте меню. Значение теста берется из файла строковых ресурсов app/res/values/strings.xml через значение идентификатора строковой константы «@string/action_settings». Для наших пунктов меню поступаем аналогично.
- **android:orderInCategory** — задает месторасположение пункта меню в списке всех пунктов меню текущей категории с помощью числового значения. Чем больше числовое значение, тем ниже будет располагаться пункт меню. Для наших пунктов меню мы задаем позицию 110 и 120 соответственно. Да-да, именно с шагом 10 — а вдруг нам захочется добавить еще один пункт меню и разместить его между этими двумя — в таком случае мы просто возьмем значение из диапазона от 110 до 120.
- **app:showAsAction** — пока просто повторим значение этого атрибута для наших пунктов меню.

ШАГ ТРЕТИЙ: напишем функционал обработки выбора добавленных пунктов меню. Для этого перейдем в файл app/java/Имя_Пакета/MainActivity.java в метод **onOptionsItemSelected**. Этот метод описывался в разделе 4.4. Напишем в этом методе такой код (Листинг 5.4).

Листинг 5.4. Обработка выбора добавленных пунктов меню

```

@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    //--- Получение идентификатора выбранного пункта меню---
    int id = item.getItemId();
}

```

```
//--- Выполнение действия для выбранного пункта меню ---
switch (id)
{
//--- Выбран пункт меню "Settings" -----
    case R.id.action_settings :
        return true; //Пункт меню обработан

//--- Выбран пункт меню "Show Hello World" -----
    case R.id.action_hello_world :
        Toast.makeText(this,
                    R.string.hello_world,
                    Toast.LENGTH_SHORT).show();
        return true; //Пункт меню обработан

//--- Выбран пункт меню "Show Android Forever" -----
    case R.id.action_android_forever :
        Toast.makeText(this,
                    R.string.android_forever,
                    Toast.LENGTH_SHORT).show();
        return true; //Пункт меню обработан
}

//--- Если не выбран ни один из наших пунктов меню --
return super.onOptionsItemSelected(item);
}
```

Все, наше первое Android приложение готово. Теперь можно переходить к запуску этого приложения в эмуляторе.

6. Запуск приложения. Эмуляторы

6.1. Эмулятор Google-Android

Теперь пришло время запустить наше первое приложение. Мы уже позаботились о создании виртуального устройства, поэтому запускаем наше приложение. Для этого или воспользуемся пунктом меню Android Studio «Run» / «Run ‘app’» (напоминаю, что наш модуль называется «app», когда вы создадите другой модуль и он будет называться например «abc», то в меню вы увидите «Run ‘abc’») или кликнем мышью по иконке на панели инструментов (см. Рис. 6.1) или нажмем горячую комбинацию клавиш «Shift+F10».

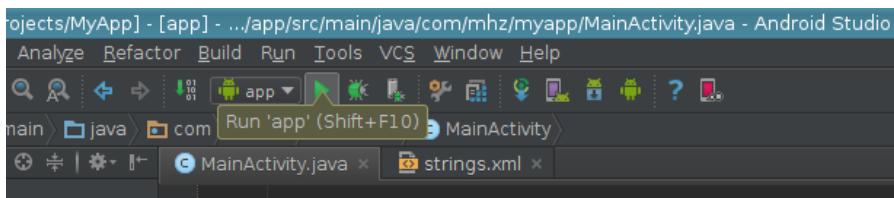


Рис. 6.1. Иконка запуска приложения
на панели задач Android Studio

При запуске приложения появится окно выбора виртуального устройства (см. Рис. 6.2).

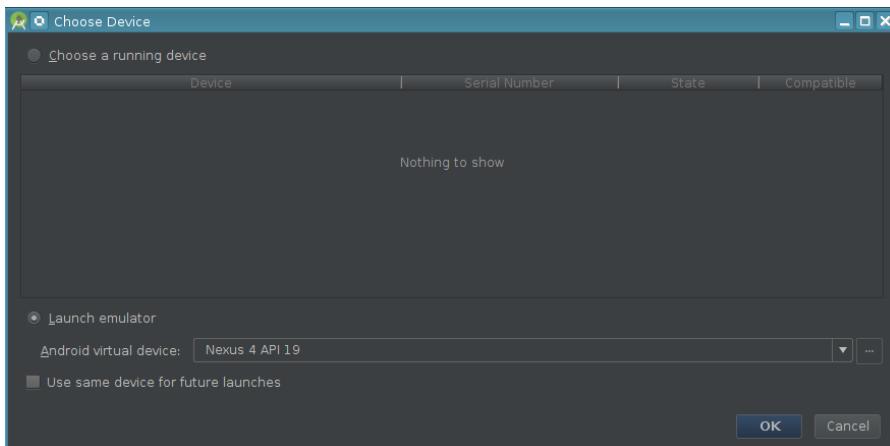


Рис. 6.2. Окно выбора виртуального устройства, в котором запустится приложение

В этом окне находятся следующие опции:

- «**Choose a running device**» — список ранее запущенных виртуальных устройств. Можно выбрать ранее запущенное устройство. Это рекомендуется, так как запуск эмулятора с виртуальным устройством занимает достаточно длительное время. На нашем Рисунке нет работающих виртуальных устройств, поэтому нам понадобится следующая опция.
- «**Launch emulator**» — опция предлагает запустить эмулятор с виртуальным устройством и выбрать виртуальное устройство из выпадающего списка. В выпадающем списке будут отображены только те виртуальные устройства, которые были созданы с помощью AVD менеджера. Виртуальные устройства других эмуляторов (например GenyMotion) в этот список попадать не будут, и поэтому должны быть запущены самостоятельно

до запуска приложения, чтобы иметь возможность их выбора из списка «Choose a running device». На данном этапе мы выберем виртуальное устройство из выпадающего списка (мы создали устройство ранее). Выбирайте устройство.

- «**Use same device for future launches**» — можно поставить галочку, чтобы при последующем запуске приложения автоматически запускать его в уже выбранном виртуальном устройстве. При этом окно выбора виртуальных устройств (Рис. 6.2) появляться при запуске приложений не будет. Удобная опция. На ваш выбор. Иногда — когда приложение тестируется в разных API эту опцию не используют. Но в большинстве случаев разработчики устанавливают галочку в этот чекбокс.

Нажимаем на кнопку «OK». Начался процесс запуска эмулятора Google-Android, который занимает достаточно длительное время, поэтому запаситесь терпением. Рекомендуется на закрывать эмулятор до завершения вашей работы в Android Studio — так как последующие запуски разрабатываемого приложения будут запускаться в уже запущенном эмуляторе.

Когда наше приложение запустится в эмуляторе, мы увидим Активность в том виде, в котором наблюдали ее в процессе верстки/разработки. Нажимаем на Меню. Появляется выпадающее меню с тремя пунктами: «Settings» (пункт был добавлен при создании модуля), «Show Hello World» и «Show Android Forever» (это наши пункты меню). Выбираем пункт меню «Show Hello World» и видим появившееся сообщение Тоста «Hello World!». Работу нашего первого приложения можно увидеть на Рис. 6.3.



Рис. 6.3. Работа нашего первого Android приложения

Предлагаю самостоятельно добавить еще один пункт меню на ваш выбор и вывести свое сообщение.

6.2. Эмулятор Genymotion

Как уже упоминалось, эмулятор Google-Android не является единственным эмулятором для запуска и тестирования разрабатываемых приложений для операционной системы Android. Высокую популярность получили эмуляторы Genymotion (<http://genymotion.com>) из-за своей высокой производительности.

В отличии от Google-Android эмуляторов, эмуляторы Genymotion не эмулируют работу Операционной Системы Android, а запускают ее на виртуальной машине VirtualBox (<http://virtualbox.org>). Решение оказалось очень удачным. Оценить производительность эмуляторов Genymotion вы сможете самостоятельно.

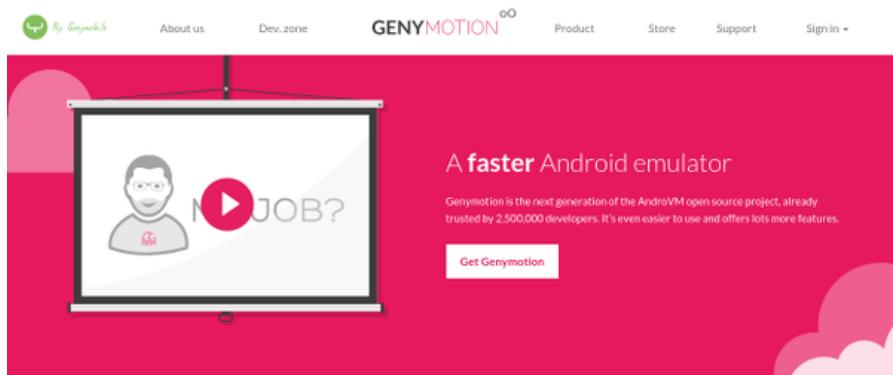


Рис. 6.4. Стартовая страница сайта компании Genymotion

В версии эмулятора для Windows включено программное обеспечение VirtualBox, поэтому при установке эмулятора на семейство ОС Windows VirtualBox будет установлен автоматически. Для пользователей семейств ОС Linux необходимо будет установить VirtualBox самостоятельно.

Заходим на сайт Genymotion (<http://genymotion.com>), делаем клик на «Get Genymotion», выбираем «Загрузить бесплатную версию» и скачиваем ее. Бесплатная версия имеет ограниченные возможности для эмулирования разрабатываемых приложений (вы сможете ознакомиться с этим списком возможностей на сайте), но этих возможностей вполне хватает для разработки большинства Android приложений. Для того чтобы зарузыть программное обеспечение Genymotion необходимо зарегистрироваться в их системе. Регистрация не сложная.

После установки запустите Genymotion. Вы увидите главное окно приложения (Рис. 6.5). В этом окне вы увидите список ранее созданных виртуальных устройств (при

первом запуске приложения вы увидите пустой список). Эти виртуальные устройства можно запускать, удалять, изменять их характеристики.

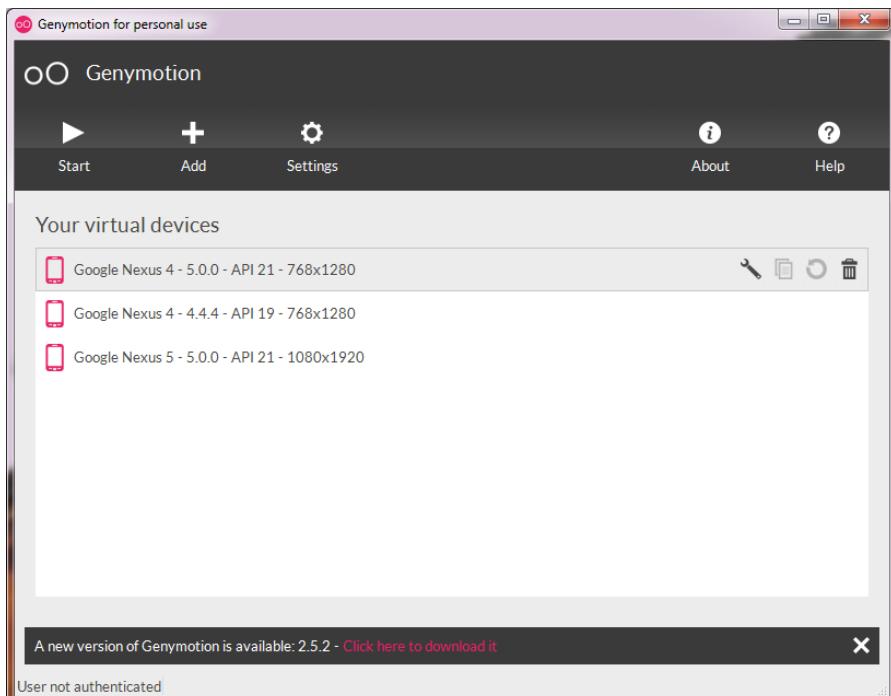


Рис. 6.5. Главное окно Genymotion

Для того чтобы добавить новое виртуальное устройство необходимо нажать на кнопку «Add». В появившемся окне после прохождения процедуры авторизации (Кнопка «SignUp») появится список доступных для установки через Интернет виртуальных устройств (см. Рис. 6.6).

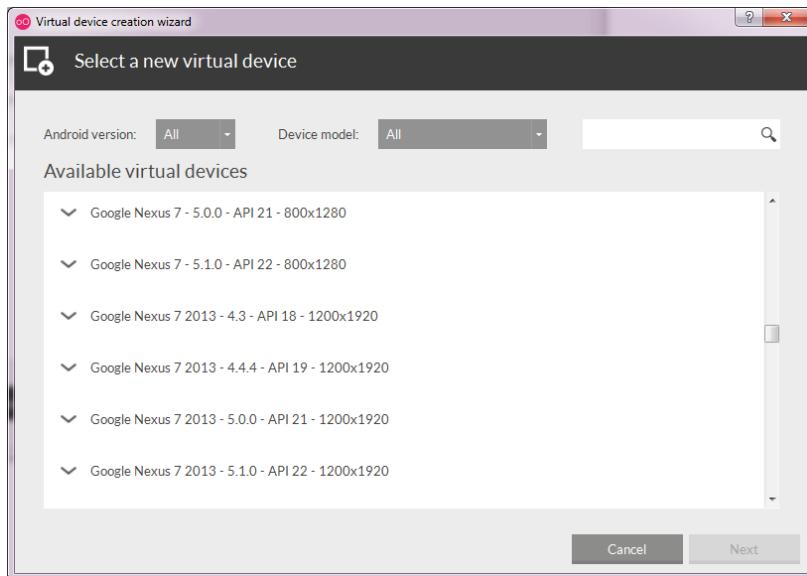


Рис. 6.6. Выбор нового виртуального устройства Genymotion

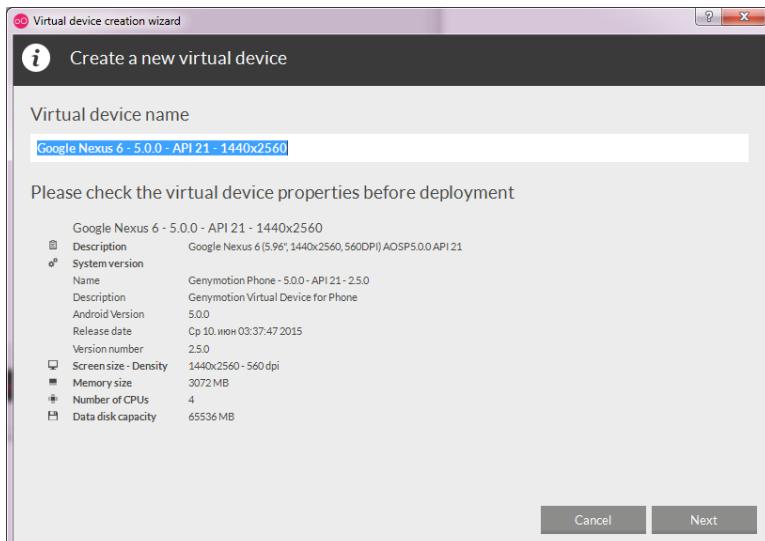


Рис. 6.7. Подробная информация о выбранном новом виртуальном устройстве

Выбираем подходящее нам виртуальное устройство, нажимаем на кнопку «Next». Появится окно с подробной информацией о виртуальном устройстве, которое мы хотим установить. Нажимаем на кнопку «Next». Начнется процесс загрузки виртуального устройства на ваш компьютер (см. Рис. 6.8).

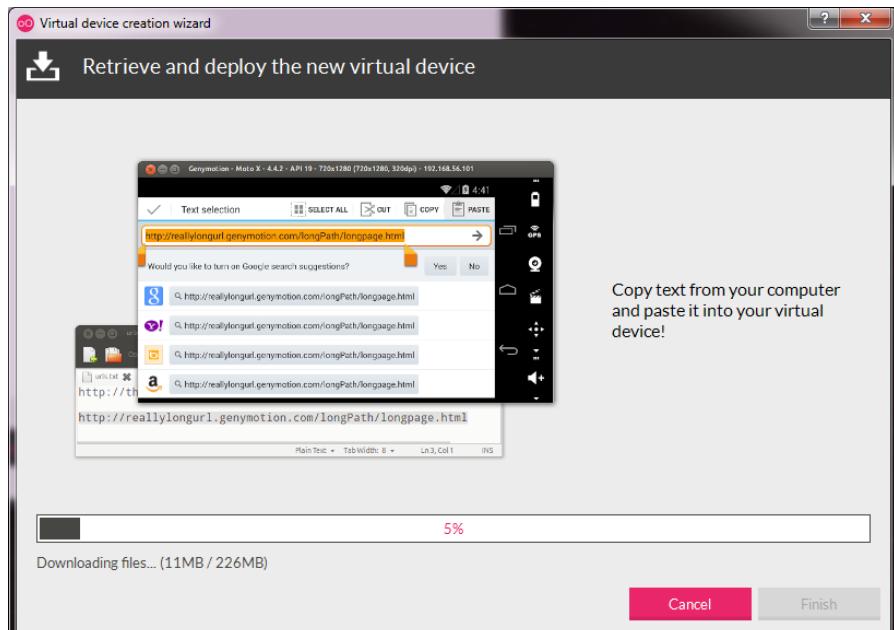


Рис. 6.8. Окно прогресса загрузки виртуального устройства

После установки вам будет доступна конфигурация виртуального устройства (см. Рис. 6.9). Рекомендую на первых этапах обучения не выделять для виртуального устройства большой размер оперативной памяти (опция «Base Memory»). Ваши приложения не будут большими, а размер памяти будет все равно выделяться на вашем компьютере.

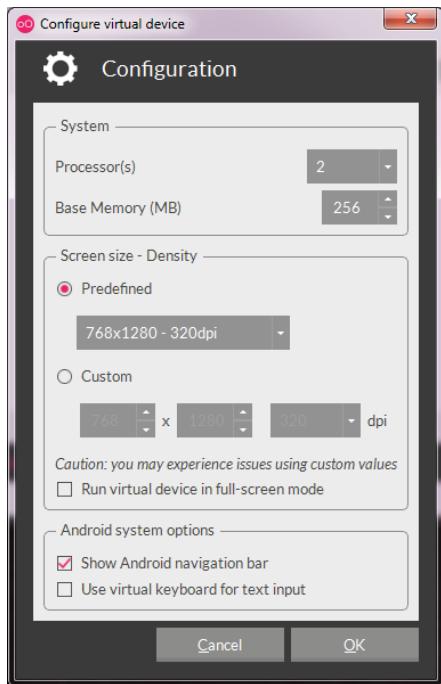


Рис. 6.9. Конфигурация виртуального устройства Genymotion

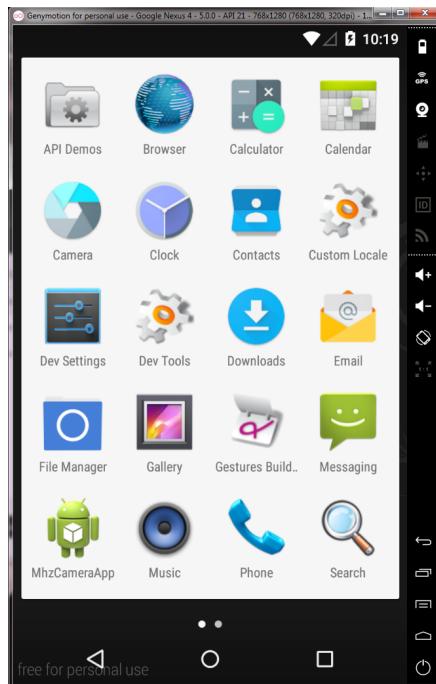


Рис. 6.10. Внешний вид эмулятора Genymotion

Сконфигурированное виртуальное устройство можно запустить, нажав на кнопку «Start» в главном окне Genymotion (см. Рис. 6.5). Появится окно эмулятора (см. Рис. 6.10). Оцените скорость загрузки эмулятора Genymotion. Она должна быть на порядок выше чем скорость загрузки эмуляторов Google-Android. Но это не означает, что какой-то эмулятор лучше или хуже. У каждого эмулятора есть свои плюсы и минусы. В процессе работы вы наберете опыт и сами сделаете свои выводы. Однако, мои рекомендации относительно эмуляторов

заключаются в следующем: используйте все доступные вам эмуляторы, делайте акцент на эмуляторе, который предоставляет вам максимум комфорта при разработке каждого отдельного вашего приложения.

Запустив выбранное виртуальное устройство Genymotion в эмуляторе, давайте посмотрим, как запустить в нем наше разрабатываемое приложение. Вернитесь в Android Studio и запустите наше приложение (меню «Run» / «Run ‘app’» или комбинация клавиш «Shift+F10»). Появится уже знакомое вам окно выбора виртуального устройства. В списке исполняемых виртуальных устройств вы увидите виртуальное устройство Genymotion (см. Рис. 6.11). Приятного вам тестирования.

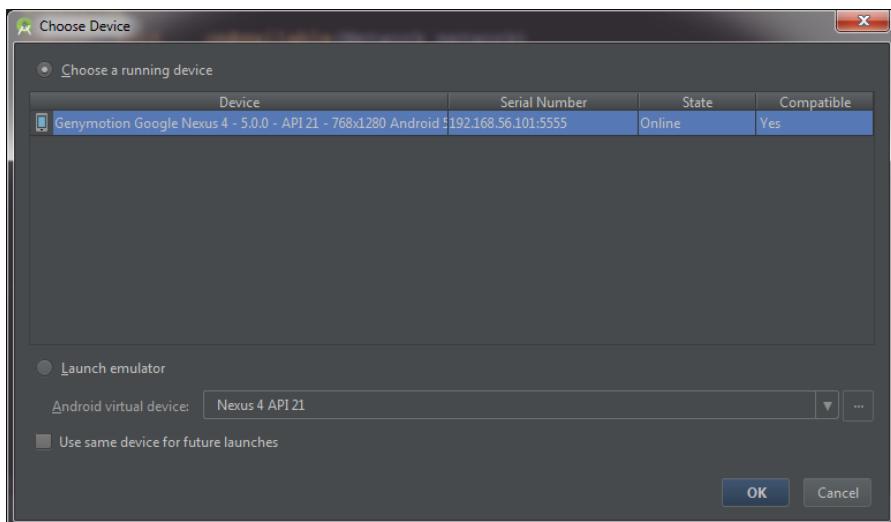


Рис. 6.11. Выбор виртуального устройства Genymotion в окне выбора устройств Android Studio при запуске приложения

6.3. Запуск приложения на реальном устройстве

Для отладки приложения на реальном устройстве, необходимо подключить устройство к компьютеру. В операционной системе Windows потребуется установка драйвера «Android Debug Bridge», который необходимо загрузить с сайта производителя этого устройства (если Windows его сама не распознает). Затем вам необходимо настроить устройство для разработки/отладки. Для этого прочитайте документацию на сайте для разработчиков: <http://developer.android.com/tools/device.html> и следуйте изложенным там инструкциям. От себя хочу добавить, что отладка Android-приложения на реальном устройстве осуществляется быстрее чем в каком-либо эмуляторе и имеет ряд других преимуществ. Однако, вовсе не нужно стремиться отлаживать ваши приложения только на реальном устройстве.

7. Логи приложения. Класс android.util.Log

Когда вы запускаете приложение для тестирования в Android Studio, вы можете видеть логи приложения (Рис. 7.1) в окне Android Studio — «Logcat».

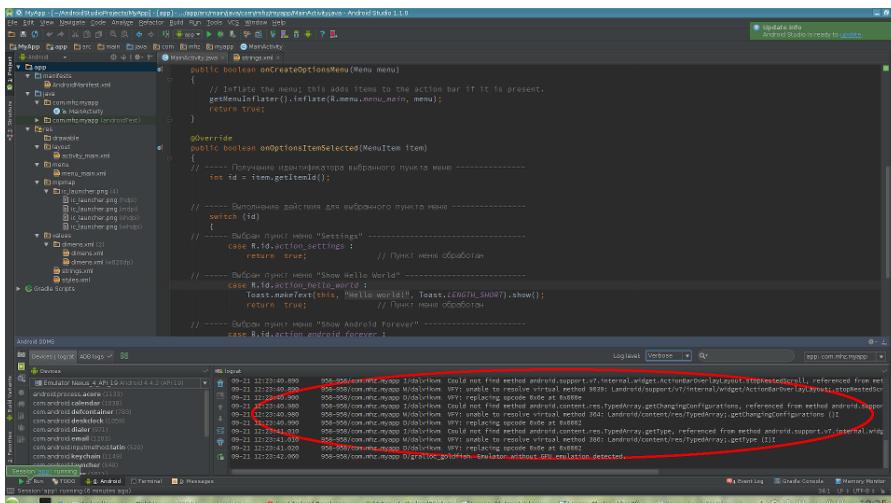


Рис. 7.1. Окно Logcat в Android Studio

У вас есть возможность добавлять свои лог-сообщения. Для этого предназначен класс android.util.Log (<http://developer.android.com/reference/android/util/Log.html>). У лог-сообщений есть несколько уровней:

- **ERROR** — Сообщения об ошибках:

```
public static int e (String tag, String msg);
```

- **WARN** — Сообщения о предупреждениях:

```
public static int w (String tag, String msg);
```

- **INFO** — Информационные сообщения:

```
public static int i (String tag, String msg);
```

- **DEBUG** — Отладочный вывод (трассировка):

```
public static int d (String tag, String msg);
```

- **VERBOSE** — Сообщения разработчиков:

```
public static int v (String tag, String msg);
```

Все эти методы принимают 2 параметра:

- **tag** — Используется для идентификации источника лог-сообщения.
- **msg** — Непосредственно само лог-сообщение.

Очень часто, для формирования лог-сообщений от Активности поступают таким образом (Листинг 7.1):

Листинг 7.1. Пример вывода лог-сообщения

```
class MainActivity    extends    Activity
{
    public    final static String TAG = "==== MainActivity";
    ...
    public    void    someMethod()
    {
        ...
        Log.d(TAG, "Лог-сообщение");
    }
}
```

Как видно из Рис. 7.1, над окном Logcat есть выпадающий список «Log Level» с помощью которого можно фильтровать сообщения по уровням.

8. Активность. События, через которые проходит Активность

Познакомившись с первыми этапами процесса разработки Android приложений перейдем непосредственно к изучению самого программирования. И начнем мы знакомство с Активности.

Класс активности является классом `android.app.Activity` или классом, производным от него (<http://developer.android.com/reference/android/app/Activity.html>). Активность, это сущность, в которой пользователь приложения может что-то делать. Активность содержит в себе контейнеры и виджеты (элементы управления). Активность имеет свой жизненный цикл, который состоит из трех основных состояний: выполнение, приостановка, остановка. Переход между этими основными состояниями сопровождается событиями. Разработчик Android приложения может создавать обработчики этих событий в классе Активности и использовать их для решения своих задач. На Рис. 8.1 представлена диаграмма переходов между состояниями Активности и события, которые генерируются при переходе из одного состояния в другое.

Как видно из Рис.8.1 у Активности есть следующие события:

- **onCreate** — активность создается.
- **onStart** — активность становится видимой для пользователя.

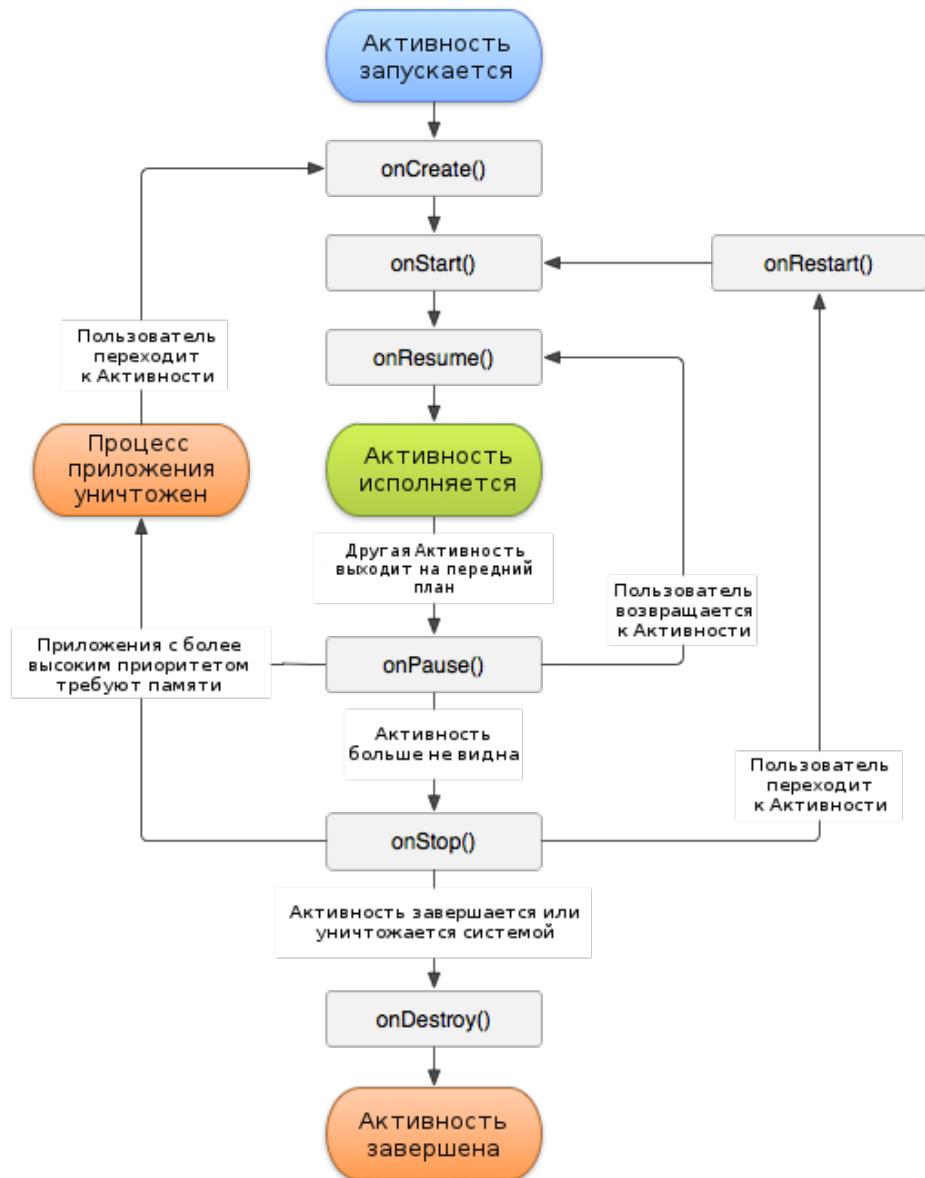


Рис. 8.1. Жизненный цикл Активности

- **onResume** — активность выходит на передний план.
- **onPause** — активность уходит с переднего плана.
- **onStop** — активность перестает быть видимой.
- **onDestroy** — активность уничтожается.

Чтобы увидеть жизненный цикл на практике, создадим новый модуль (app2) и переопределим в файле MainActitity в классе Активности эти методы таким образом, чтобы добавить в них вывод лог-сообщений (Листинг 8.2).

Листинг 8.1. Обработка событий жизненного цикла Активности

```
public class MainActivity extends ActionBarActivity
{
    //---- константы класса -----
    public final static String TAG = "==>MainActivity";

    //---- Методы класса -----
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    protected void onStart()
    {
        super.onStart(); //Обязательно вызвать
                        //метод родительского
                        //класса
        Log.d(TAG, "onStart");
    }
}
```

```
@Override  
protected void    onResume ()  
{  
    super.onResume (); //Обязательно вызвать  
                      //метод родительского  
                      //класса  
    Log.d(TAG, "onResume");  
}  
  
@Override  
protected void    onPause ()  
{  
    super.onPause (); //Обязательно вызвать  
                      //метод родительского  
                      //класса  
    Log.d(TAG, "onPause");  
}  
  
@Override  
protected void    onStop ()  
{  
    super.onStop (); //Обязательно вызвать  
                      //метод родительского  
                      //класса  
    Log.d(TAG, "onStop");  
}  
  
@Override  
protected void    onDestroy ()  
{  
    super.onDestroy (); //Обязательно вызвать  
                      //метод родительского  
                      //класса  
    Log.d(TAG, "onDestroy");  
}  
  
...  
}
```

Запускайте пример (Листинг 8.1) и поэкспериментируйте с Активностью. На каждое ваше действие с Активностью — поворот устройства, нажимание на кнопку «Назад», запуск другого приложения — вы будете видеть лог-сообщения, выводимыми в обработчиках событий жизненного цикла.

Обратите внимание, что при повороте устройства Активность создается заново. Так же, операционная система Android уничтожает текущую Активность и создает новую при каждом изменении конфигурации времени выполнения (Поворот устройства так же относится к изменению конфигурации времени выполнения). Поскольку Активность представлена объектом, то при пересоздании Активности происходит уничтожение старого объекта Активности, и следовательно — все значения полей этого объекта уничтожаются, а в новом объекте Активности значения полей объекта принимают начальные значения. Это неприятный факт, но с ним нужно смириться и обязательно учитывать его при разработке своих приложений. Android предоставляет механизмы сохранения данных при изменении конфигурации времени выполнения. Об этих механизмах будет рассказано в следующих уроках. А пока всего лишь сделаем упоминание о том, что одним из механизмов сохранения данных является объект **Bundle**. Обратите внимание, что метод `onCreate` принимает параметр `savedInstanceState` типа **Bundle**. Этот параметр не будет равен `null` если объект Активности перед уничтожением сохранил какие-то данные в этом **Bundle** объекте. Для того, чтобы Активность имела возможность сохранить данные в объект **Bundle** перед приостановкой, необходимо создать обработчик события `onSaveInstanceState`:

```
protected void onSaveInstanceState(Bundle bundle);
```

путем переопределения этого метода в классе Активности. Этот метод принимает объект **Bundle**, в который и нужно выполнить сохранение нужных данных. Ранее сохраненные данные будут переданы в виде того же объекта **Bundle** в метод **onCreate**. В Листинге 8.2 показан простейший пример сохранения данных и восстановления этих данных в методе **onCreate** при создании Активности. Но более подробная информация о механизмах сохранения данных при пересоздании Активностей будет рассказано в одном из следующих уроков.

Листинг 8.2. Пример сохранения/восстановления данных в Bundle

```
public class MainActivity extends ActionBarActivity
{
    //--- константы класса -----
    public final static String TAG = "==>MainActivity";

    /**
     * Ключ для значения, сохраняемого в Bundle
     */
    private final static String KEY_RANDOM_VALUE =
            "RandomValue";

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

        TextView TV = (TextView) this.findViewById(R.id.tv1);
```

```

        if (savedInstanceState != null)
        {
    //--- Восстанавливаем значение текстового поля ---
            String value = savedInstanceState.
                getString(MainActivity.KEY_RANDOM_VALUE);
            TV.setText(value);
        }
        else
        {
    //--- Приложение запускается впервые — запишем
    //что-то в текстовое поле
            int value = (int)(Math.random() * 100);
            TV.setText("Случайное число: " + value);
        }
    }

@Override
public void onSaveInstanceState(Bundle bundle)
{
    super.onSaveInstanceState(bundle);
    Log.d(TAG, "onSaveInstanceState");

    //--- Читаем значение из Текстового поля -----
    TextView TV = (TextView) this.
                    findViewById(R.id.tv1);
    String value = TV.getText().toString();

    //--- И помещаем его в Bundle -----
    bundle.putString(MainActivity.
                    KEY_RANDOM_VALUE, value);
}

```

Что для нас оказалось новым в Листинге 8.2 — вызов метода **findViewById**. Давайте познакомимся с этим методом, поскольку на протяжении всего курса этот метод придется вызывать чаще всего:

```
public View findViewById (int id);
```

Метод является методом объекта Активности (класс **android.app.Activity**). Предназначен для нахождения виджета по его идентификатору. Принимаемое значение — идентификатор виджета или контейнера. Возвращаемое значение — ссылка на объект **android.view.View** (родительский класс всех виджетов) или null если виджет не найден. Откуда взялось значение идентификатора текстового поля **R.id.tv1** в листинге 8.2? Это значение необходимо задать самостоятельно в файле макета Активности **app2/res/layout/activity_main.xml** (см. Листинг 8.3). Обратите внимание, каким образом задаются идентификаторы в xml-файле макета Активности — «@+id/значение_идентификатора». Значение идентификатора придумывает разработчик Android приложения. После этого, значение идентификатора с Java-исходном-коде становится доступным через класс R в виде «R.id.значение_идентификатора» (см. Листинг 8.2).

Листинг 8.3. Назначение идентификатора для виджета TextView

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/tv1"  
/>
```

9. Верстка макета Активности с помощью XML

Как вы уже успели заметить из вышеизложенного, создание макета Активности осуществляется с помощью xml разметки в файле **имя_модуля/res/layout/activity_main.xml**. В этом разделе будет рассказано о принципах верстки макетов, а вся остальная информация будет предоставлена во всех последующих уроках данного курса.

На Листинге 9.1 можно увидеть концепцию верстки макета.

Листинг 9.1. Концепция верстки макета Активности

```
<главный_контейнер
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MainActivity"
    другие_атрибуты>

    <виджет_или_контейнер_1
        android:layout_width="..."
        android:layout_height="..."
        android:layout_gravity="..."
        другие_атрибуты>
        <виджет_или_контейнер_1_1
            android:layout_width="..."
            android:layout_height="..."
            android:layout_gravity="..."
            другие_атрибуты>
    </виджет_или_контейнер_1_1>
```

```
<виджет_или_контейнер_1_2  
    android:layout_width="..."  
    android:layout_height="..."  
    android:layout_gravity="..."  
    другие_атрибуты>  
    </виджет_или_контейнер_1_2>  
    ...  
</виджет_или_контейнер_1>  
  
<виджет_или_контейнер_2  
    android:layout_width="..."  
    android:layout_height="..."  
    android:layout_gravity="..."  
    другие_атрибуты>  
    ...  
</виджет_или_контейнер_2>  
...  
</главный_контейнер>
```

Как видно из Листинга 9.1 корневым элементом макета Активности является главный контейнер. Конечно, можно в качестве корневого элемента разместить не контейнер, а например виджет (элемент управления) **Button** и ничего страшного при этом не произойдет, но все же рекомендуемым корневым элементом является контейнер. Контейнер по совместительству является еще и менеджером раскладки. Поэтому выбор контейнера/менеджера_раскладки непосредственно определяется разработчиком в зависимости от правил размещения виджетов на Активности. Обязательными атрибутами корневого (главного) контейнера являются атрибуты **xmlns:android="http://schemas.android.com/apk/res/android"** и **xmlns:tools="http://schemas.android.com/tools"**, которые подключают XML-схемы с правилами составления

макета и назначают элементам этим схем пространства имен (**android** и **tools**). Кроме этого, в главном контейнере могут находиться и другие атрибуты, например атрибуты отступа содержимого контейнера от границ контейнера (см. Листинг 4.3).

Внутри главного контейнера могут располагаться или виджеты или другие контейнеры. Для всех вложенных виджетов или контейнеров обязательными являются атрибуты **android:layout_width** и **android:layout_height**. Эти атрибуты задают соответственно ширину и высоту виджета/контейнера. Этим атрибутам могут быть назначены значения **match_parent** (занять всю область, предоставляемую родителем) или **wrap_content** (занять область, равную размеру своего содержимого) или фиксированное числовое значение с суффиксом единицы измерения (например px, dp, и т. д.). Необязательным атрибутом, но не редко используемым, является атрибут **android:layout_gravity** — он задает куда будет «причаливать» («притягиваться») виджет, в случае, если размер виджета меньше области, предоставляемой родительским контейнером:

- **center_horizontal** — центрировать по горизонтали;
- **center_vertical** — центрировать по вертикали;
- **clip_horizontal** — отсечь по горизонтали;
- **clip_vertical** — отсечь по вертикали;
- **fill_horizontal** — растянуть по горизонтали;
- **fill_vertical** — растянуть по вертикали;
- **bottom** — разместить внизу;
- **center** — разместить по центру;

- **left** — разместить слева;
- **right** — разместить справа;
- **top** — разместить сверху.

Значения этого атрибута можно комбинировать, например:

```
android:layout_gravity="center_horizontal | top"
```

Под «другие атрибуты» могут подразумеваться остальные атрибуты, как общие для всех, так и специфические для каждого конкретного виджета или контейнера. Например атрибут **android:id** — задает уникальный идентификатор виджета/контейнера. Или еще, например, атрибут **android:onClick** — задает имя метода обработчика события клика (нажатия) по виджету/контейнеру.

Отдельно необходимо отметить то, что названия xml элементов для верстки сопадают с названием классов в Java для соответствующих объектов. Например, элемент `<TextView .../>` имеет соответствующий класс **android.widget.TextView** и т.д.

10. Базовые виджеты: TextView, Button, EditText

Внешний вид виджетов, создаваемых и рассматриваемых в этой главе можно видеть на Рис. 10.1. Исходный код — в модуле app2.



Рис. 10.1. Базовые виджеты TextView, Button, EditText

10.1. TextView

Виджет **TextView** отображает текст для пользователя и при дополнительных настройках, может позволить редактировать этот текст.

Иерархия классов для **TextView**:

```
android.lang.Object
|
+--- android.view.View
|
+--- android.widget.TextView
```

Класс является наследником класса **android.view.View**. Полное описание полей и методов класса можно прочитать на сайте для разработчиков по адресу: <http://developer.android.com/reference/android/widget/TextView.html>.

Виджет **TextView** создается с помощью xml верстки следующим образом (Листинг 10.1):

Листинг 10.1. Пример создания виджета **TextView** в xml-макете Активности

```
<TextView
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/tv1"
    android:layout_gravity="center_horizontal"
    android:textSize="32sp"
    android:textColor="#330066"
    android:textStyle="bold"
/>
```

На Листинге 10.1 выделены жирным несколько новых для вас атрибутов — **android:textSize**, **android:textColor**,

android:textStyle. Названия атрибутов говорят сами за себя — поэтому можете смело ими пользоваться. Обратите внимание, что единицы измерения для размера текста нужно использовать «sp» вместо «dp» («sp» это то же самое что и «dp» только для текста).

Из методов класса отметим в этом уроке (остальное нужно читать на сайте для разработчиков) следующие:

- **void setText(CharSequence text)** — записывает текст в текстовое поле TextView;
- **void setText(int resId)** — записывает текст в виджет. Сам текст в метод передается в виде идентификатора строки которая находится в файле ресурсов: имя_модуля/res/values/strings.xml;
- **CharSequence getText()** — метод считывает строковое содержимое из виджета TextView.

10.2. Button

Виджет **Button** предстает из себя «push-button» (нажимаемую кнопку), по которой пользователь может осуществить клик или нажатие для выполнения действия.

Иерархия классов для **Button**:

```
android.lang.Object
|
+--- android.view.View
|
+--- android.widget.TextView
|
+--- android.widget.Button
```

Полное описание полей и методов класса можно прочитать на сайте для разработчиков по адресу: <http://developer.android.com/reference/android/widget/Button.html>.

Виджет **Button** создается с помощью xml верстки следующим образом (Листинг 10.2):

Листинг 10.2. Пример создания виджета Button в xml-макете Активности

```
<Button  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="@string/click_me"  
    android:id="@+id	btn1"  
    android:textSize="20sp"  
    android:layout_marginTop="20dp"  
    android:onClick="btnClick"  
/>
```

На Листинге 10.2 жирным выделены новые для вас атрибуты:

- **android:layout_margin** — внешние отступы до соседних виджетов: **layout_marginTop**, **layout_marginBottom**, **layout_marginLeft**, **layout_marginRight**;
- **android:onClick** — обработчик события клика по кнопке, **public** метод в классе Активности (метод создается программистом вручную). Пример метода обрабатывающего событие клика показан в Листинге 10.3. Сразу отмечается, что это не единственный способ назначения виджету **Button** обработчика события. Обработчики событий можно назначать и из исходного кода Java — об этом будет описано в соответствующей главе этого урока.

Листинг 10.3. Пример обработчика события клика по кнопке

```
public class MainActivity extends ActionBarActivity
{
    ...
    /**
     * Обработчик события клика по кнопке или другому виджету
     * @param view - Виджет, по которому осуществлен клик
     */
    public void btnClick(View view)
    {
        int id = view.getId();
        switch (id)
        {
            case R.id.btn1 :
                Toast.makeText(
                    this,
                    "Button Clicked!",
                    Toast.LENGTH_SHORT).show();
                break;
        }
    }
}
```

Еще хочется отметить, что методы **setText/getText** работают для виджета **Button** аналогичным образом как и для виджета **TextView**.

10.3. EditText

Виджет **EditText** представляет из себя «тонко настроенный» **TextView** для редактирования текста.

Иерархия классов для **EditText**:

```
android.lang.Object
|
+--- android.view.View
|
+--- android.widget.TextView
|
+--- android.widget.
    EditText
```

Полное описание полей и методов класса можно прочитать на сайте для разработчиков: <http://developer.android.com/reference/android/widget/EditText.html>.

Виджет **EditText** создается с помощью xml верстки следующим образом (Листинг 10.4):

Листинг 10.4. Создание виджета EditText
в xml-макете Активности

```
<EditText
    android:text="@string/hello_world"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/edt1"
    android:textSize="30sp"
    android:layout_marginTop="20dp"
/>
```

11. События клика, перемещения при касании

Для примера обработки событий клика и перемещения при касании, создадим новый модуль — «app3». В рассматриваемом примере события клика и перемещения будут обрабатываться для главного контейнера Активности. Для вывода информации о каждом событии в макете Активности создадим два виджета **TextView** (см. Листинг 11.1) с идентификаторами **tvClickInfo** (будет выводится информация о кликах) и **tvMotionInfo** (будет выводится информация о перемещениях).

Листинг 11.1 Макет Активности примера для исследования событий клика и перемещения

```
<LinearLayout  
    xmlns:android=  
        "http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
  
    android:paddingLeft="@dimen/activity_horizontal_margin"  
    android:paddingRight="@dimen/activity_horizontal_margin"  
    android:paddingTop="@dimen/activity_vertical_margin"  
    android:paddingBottom="@dimen/activity_vertical_margin"  
  
    tools:context=".MainActivity"  
  
    android:orientation="vertical"  
    android:id="@+id/l11"  
>
```

```

<TextView
    android:text="@string/click_info"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:id="@+id/tvClickInfo"
    android:textSize="10pt"
/>

<Space
    android:layout_width="match_parent"
    android:layout_height="20dp"
/>

<TextView
    android:text="@string/motion_info"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:id="@+id/tvMotionInfo"
    android:textSize="10pt"
/>
</LinearLayout>
```

Для того, чтобы назначить виджету или контейнеру обработчик события клика, необходимо использовать метод, объявленный в **android.view.View**:

```
public void setOnClickListener (View.OnClickListener l);
```

Метод **setOnClickListener** принимает ссылку, на объект, реализующий интерфейс **View.OnClickListener**, в котором объявлен всего лишь один метод:

```
public abstract void onClick (View v);
```

где **v** — вид (виджет или контейнер), по которому осуществлен клик.

Например (Листинг 11.2):

Листинг 11.2 Пример назначения обработчика события клика

```
Button B = (Button) this.findViewById(R.id.  
        идентификатор_кнопки);  
B.setOnClickListener(  
    new View.OnClickListener()  
    {  
        @Override  
        public void onClick(View v)  
        {  
            ... // Код обработки события клика по виджету v  
        }  
    } );
```

Для того, чтобы виджету или контейнеру назначить обработчик события перемещения, необходимо воспользоваться методом, объявленным в **android.view.View**:

```
public void setOnTouchListener (View.OnTouchListener l);
```

Этот метод принимает ссылку на объект, реализующий интерфейс **View.OnTouchListener**, в котором объявлен один метод обработчик события:

```
public abstract boolean onTouch (View v,  
        MotionEvent event);
```

Метод, обрабатывающий событие, принимает следующие параметры:

- **View v** — виджет, источник события.
- **MotionEvent event** — объект, содержащий информацию о событии.

Пример назначения обработчика событий касания и перемещения (Листинг 11.3):

Листинг 11.3. Пример назначения обработчика событий касания и перемещения

```
LinearLayout LL = (LinearLayout)
    this.findViewById(
        R.id.идентификатор_контейнера);
LL.setOnTouchListener(
    new View.OnTouchListener()
    {
        @Override
        public boolean onTouch(View v,
                              MotionEvent event)
        {
            ... // Код обработки событий касания
        }
    });
}
```

Назначая обработчик события **onTouchListener**, у нас появляется возможность получать сообщения не только о событиях перемещения, но и о событиях нажатия (касания) и отпускания. Событие перемещения представляется объектом **android.view.MotionEvent** (<http://developer.android.com/reference/android/view/MotionEvent.html>) в котором предоставляется инструментарий для получения всевозможных данных о событиях. Этот объект так же позволяет получить информацию о множественных касаниях и перемещениях (это когда пользователь использует в касании несколько пальцев одновременно — например делает движение увеличения изображения). Отдельно обратим внимание на следующие методы класса **android.view.MotionEvent**:

- **public final int getAction ()** — возвращает числовое значение, идентифицирующее действие пользователя (MotionEvent.ACTION_DOWN — касание, MotionEvent.ACTION_MOVE — перемещение, MotionEvent.ACTION_UP — отпускание и т. д.).
- **public final float getX()** — возвращает X-координату первого указателя (пальца пользователя).
- **public final float getY()** — возвращает Y-координату первого указателя.
- **public final int getPointerCount()** — возвращает количество указателей (пальцев пользователя) участвующих в событии.
- **public final float getX(int pointerIndex)** — возвращает X-координату указателя, индекс которого задается в параметре pointerIndex.
- **public final float getY(int pointerIndex)** — возвращает Y-координату указателя, индекс которого задается в параметре pointerIndex.
- **public final float getPressure (int pointerIndex)** — возвращает силу давления указателя, индекс которого задается в параметре pointerIndex.

В модуле «app3» реализован следующий пример (Листинг 11.4).

Листинг 11.4. Пример обработки событий клика и перемещения при касании в модуле «app3»

```
public class MainActivity extends Action
{
    //--- Class members -----
```

```
/***
 * Виджет TextView для вывода информации о клике
 */
private TextView tvClickInfo;

/***
 * Виджет TextView для вывода информации о перемещении
 */
private TextView tvMotionInfo;

/***
 * Счетчик количества кликов
 */
private int cntClick;

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

//--- Инициализируем поля объекта -----
    this.tvClickInfo =
        (TextView) this.findViewById(R.id.tvClickInfo);
    this.tvMotionInfo =
        (TextView) this.findViewById(R.id.tvMotionInfo);

//--- Находим главный контейнер в макете Активности -
    LinearLayout LL = (LinearLayout) this.
        findViewById(R.id.ll1);
//--- Назначем обработчик события "Прикосновения/
//Перемещения" -----
    LL.setOnTouchListener(new View.OnTouchListener()
    {
        @Override
```

```
public boolean onTouch(View v, MotionEvent event)
{
    //--- Получение типа события -----
    int     action    = event.getAction();

    //--- Получение координат касания -----
    float   x         = event.getX();
    float   y         = event.getY();

    //--- Формирование строки с информацией о событии --
    String S          = "";

    switch (action)
    {
        case MotionEvent.ACTION_DOWN :
            S      += "Нажатие\n";
            break;

        case MotionEvent.ACTION_MOVE:
            S      += "Перемещение\n";
            break;

        case MotionEvent.ACTION_UP :
            S      += "Отпускание\n";
            break;
    }

    S  += "X = " + x + "\nY = " + y;

    //--- Вывод информации о событии -----
    MainActivity.this.tvMotionInfo.setText(S);

    return false;
}
};

//--- Назначаем обработчик клика -----
```

```
//--- Назначаем обработчик клика -----
LL.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
//--- Увеличиваем счетчик количества кликов по Активности -
        MainActivity.this.cntClick++;

//--- Отображаем информацию в виджете tvClickInfo --
        MainActivity.this.tvClickInfo.setText(
                "Количество кликов: " +
                MainActivity.this.cntClick);
    }
});
```

...
}

Полный код примера находится в папке с исходными кодами данного урока. Внешний вид работы примера изображен на Рис. 11.1.

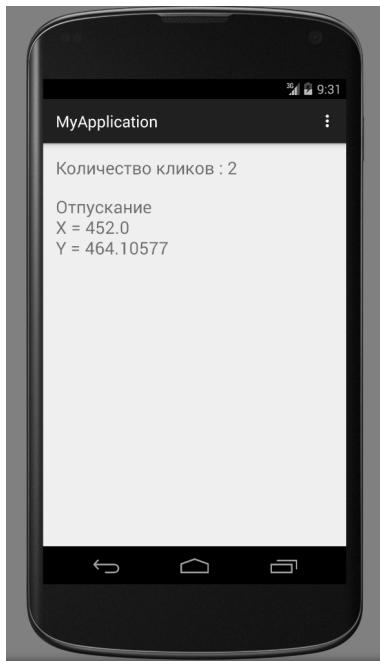


Рис. 11.1. Обработка событий клика и перемещения в приложении «app3»

12. Менеджеры раскладки LinearLayout, TableLayout

12.1. Понятие менеджера раскладки (Layout Manager)

Менеджер раскладки, это контейнер, предназначенный для содержания в себе дочерних виджетов или других контейнеров, упорядочивая их по определенным правилам. Менеджер раскладки берет на себя функционал упорядочивания дочерних виджетов, избавляя разработчиков от забот по размещению и выравниванию виджетов на экранах разного размера и ориентации.

Все менеджеры компоновки являются классами, производными от `android.view.View`, что дает возможность менеджерам компоновки размещаться внутри других менеджеров компоновок для формирования сложных раскладок путем их комбинации друг с другом.

Вот несколько классов менеджеров раскладок, с которыми мы постепенно познакомимся:

- `android.widget.LinearLayout` — менеджер раскладки виджетов по вертикали (в один столбец) или по горизонтали (в одну строку).
- `android.widget.TableLayout` — менеджер раскладки виджетов по строкам и столбцам.
- `android.widget.RelativeLayout` — менеджер раскладки виджетов относительно друг друга.

- **android.widget.GridLayout** — менеджер раскладки виджетов по ячейкам решетки (сетки).
- **android.widget.FrameLayout** — менеджер раскладки, предназначенный для размещения одиночного виджета.
- **android.widget.SccrollView** — менеджер раскладки, располагающий виджеты последовательно друг за другом в виде списка с возможностью прокрутки. Удобен для случая, если виджеты, размещенные на макете, по своим размерам не умещаются на экране устройства.

12.2 LinearLayout

Менеджер раскладки **android.widget.LinearLayout** упорядочивает свои дочерние виджеты в один столбец (вертикальная ориентация) или в одну строку (горизонтальная ориентация). Полное описание класса можно прочитать по ссылке <http://developer.android.com/reference/android/widget/LinearLayout.html>.

Иерархия классов:

```
java.lang.Object
  |
  +--- android.view.View
    |
    +--- android.view.ViewGroup
      |
      +--- android.widget.LinearLayout
```

Пример xml верстки для создания менеджера раскладки **android.widget.LinearLayout** находится в Листинге 12.1.

Листинг 12.1. Пример xml-верстки для создания менеджера раскладки LinearLayout

```
<LinearLayout  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:orientation="vertical"  
    android:id="@+id/l12">  
    ...  
</LinearLayout>
```

Ориентация дочерних виджетов задается с помощью атрибута **android:orientation**, который может принимать или значение **vertical** (вертикальная ориентация) или значение **horizontal** (горизонтальная ориентация). По умолчанию для **android.widget.LinearLayout** установлена горизонтальная ориентация. Программно можно установить ориентацию с помощью вызова метода **setOrientation()**.

Растягивание менеджера раскладки **android.widget.LinearLayout** относительно родительской области, в которой он располагается, задается с помощью атрибутов **android:layout_width** и **android:layout_height**, которые уже описывались в этом уроке.

Для выравнивания дочерних элементов, необходимо использовать атрибут **android:gravity**:

- **center, center_vertical, center_horizontal** — дочерние виджеты выравниваются по центру, по центру вертикали, по центру горизонтали соответственно.
- **top, bottom, right, left** — дочерние виджеты выравниваются к верху, к низу, справа, слева соответственно.

Чтобы задать программно выравнивание дочерних виджетов необходимо воспользоваться функцией `setGravity()`.

Рассмотрим несколько примеров верстки с помощью `android.widget.LinearLayout`. На Листинге 12.2 представлена верстка макета Активности с главным контейнером `android.widget.LinearLayout` (файл `имя_модуля/res/layout/activity_main.xml`).

Листинг 12.2. Пример упорядочивания дочерних виджетов с помощью `LinearLayout`

```
<LinearLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button1"
        />
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button2"
        />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button3"
        />
</LinearLayout>
```



Рис. 12.1. Внешний вид макета из Листинга 12.2

Как видно из Листинга 12.2 внутри менеджера раскладки `android.widget.LinearLayout` расположены три кнопки. Каждая кнопка по ширине и высоте занимает область, соответствующую размеру своего содержимого (значение `wrap_content` для высоты и ширины). Менеджеру раскладки задана вертикальная ориентация. Как это выглядит на экране устройства показано на Рис. 12.1.

Давайте поэкспериментируем с атрибутом `android:gravity` для `android.widget.LinearLayout`. Добавим этот атрибут и установим ему по очереди значения `center_horizontal`, `center_vertical`, `center`, `right`. Смотрите на

Листинг 12.3 и на то, как это выглядит на экране устройства Рис. 12.2.

Листинг 12.3. Экспериментирование со значениями атрибута android:gravity

```
<LinearLayout  
    xmlns:android=  
        "http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
  
    android:orientation="vertical"  
    android:gravity="center_horizontal"  
    >  
    ...  
</LinearLayout>
```

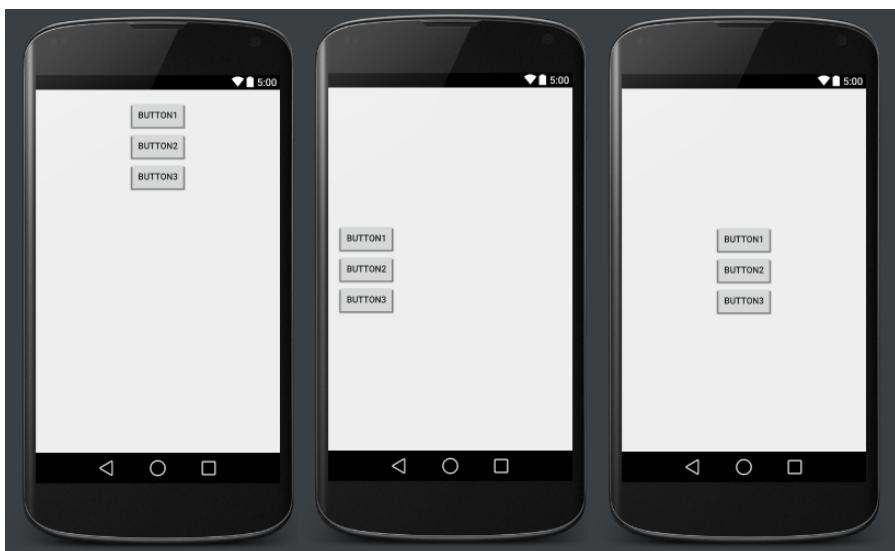


Рис. 12.2. Отображение дочерних виджетов со значениями атрибута android:gravity center_horizontal, center_vertical, center соответственно

Теперь мы получили четкое представление о том, как работает атрибут **android:gravity**. Важно отметить, что этот атрибут не является атрибутом только для **android.widget.LinearLayout**. Этот атрибут присутствует и в других контейнерах и виджетах. Например, для виджета **Button** вы можете с помощью этого атрибута управлять размещением текста на кнопке. Поэкспериментируйте самостоятельно с виджетом **Button** и его текстом 😊

Рассмотрим еще один достаточно интересный атрибут **android:layout_weight**. Этот атрибут не относится только к **android.widget.LinearLayout**. Этот атрибут можно использовать для любого дочернего виджета. В нашем примере мы будем использовать этот атрибут для дочерних виджетов — кнопок. Атрибут **android:layout_weight** управляет механизмом заполнения дочерними виджетами «оставшейся области» родительского контейнера. Атрибуту присваивается числовое значение — сколько частей от оставшейся области родительского контейнера нужно отдать виджету. Если одному виджету установить значение этого атрибута в «1», то это будет означать, что виджет займет всю оставшуюся область. Если двум виджетам присвоить значение этого атрибута в «1», то они оба займут оставшуюся область и поделят ее поровну. Начинаете понимать? Отлично!

Чтобы не объяснять долго на словах, давайте увидим этот атрибут в действии. Уберем атрибут **android:gravity** из менеджера раскладки **android.widget.LinearLayout** (чтобы не мешал) и добавим атрибут **android:layout_weight** в кнопку с текстом «Button1» (см. Листинг 12.4).

Листинг 12.4. Добавляем атрибут android:layout_weight в виджет Button

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Button1"  
    android:layout_weight="1"  
/>
```

Результат, к которому привели изменения, указанные в Листинге 12.4 изображен на Рис. 12.3. По очереди добавим этот атрибут аналогичным образом во все остальные кнопки. Результат изменения смотрите на Рис. 12.3.

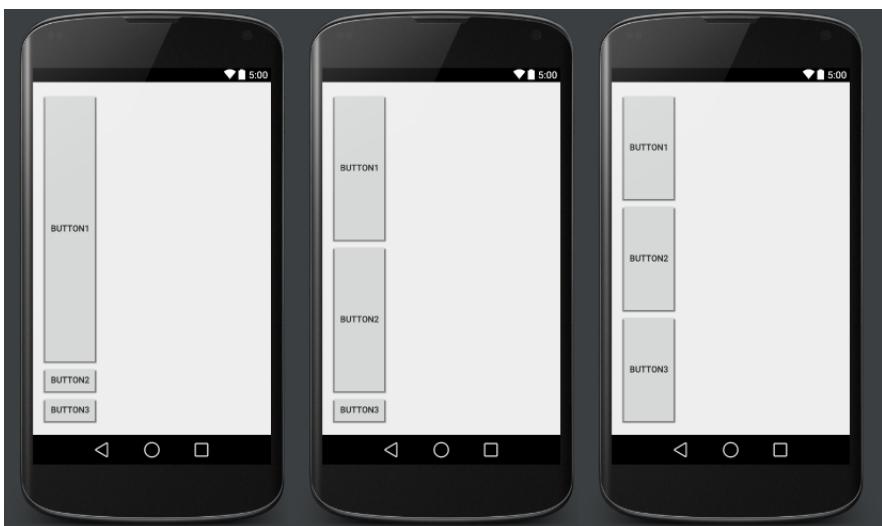


Рис. 12.3. Последовательное добавление атрибута android:layout_weight сначала в первую кнопку, потом во вторую, потом в третью

Многим начинающим разработчикам нравится атрибут android:layout_weight, однако злоупотреблять его частым использованием не нужно.

Последний штрих к примеру Листинга 12.2 — изменение значение атрибута **android:layout_width** для второй кнопки (см. Листинг 12.5).

Листинг 12.5. Другое значение атрибута **android:layout_width** для второй кнопки

```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Button2"
    android:layout_weight="1"
/>>
```

Далее, добавим атрибут **android:layout_gravity** со значением **center_horizontal** для первой кнопки (см. Листинг 12.6).

Листинг 12.6. Добавление атрибута **android:layout_gravity** для первой кнопки

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button1"
    android:layout_weight="1"
    android:layout_gravity="center_horizontal"
/>>
```

И последний штрих — добавим в третью кнопку атрибут **android:layout_gravity** со значением **right** (см. Листинг 12.7).

Листинг 12.7. Добавление атрибута **android:layout_gravity** для третьей кнопки

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

```
    android:layout_height="wrap_content"
    android:text="Button3"
    android:layout_weight="1"
    android:layout_gravity="right"
/>
```

Результат последовательных изменений из Листингов 12.5, 12.6, 12.7 можно увидеть на Рис. 12.4.

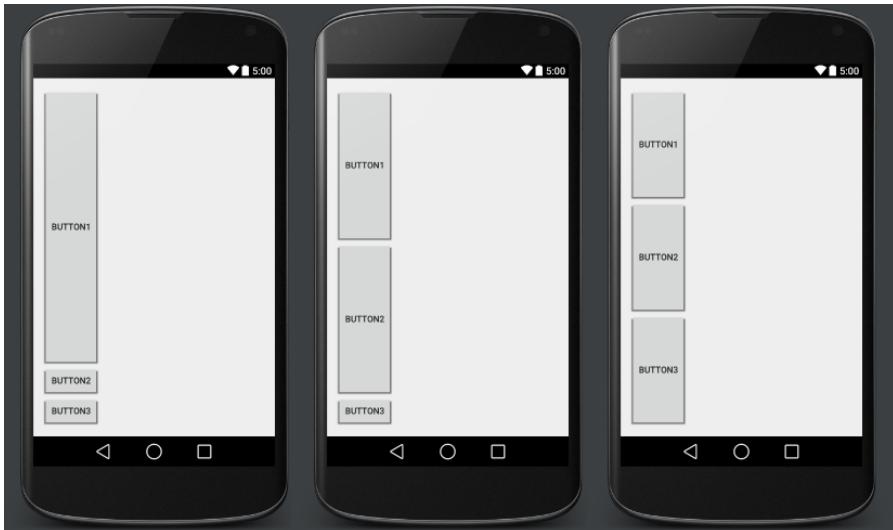


Рис. 12.4. Последовательные изменения
из Листингов 12.5, 12.6, 12.7

И последний пример для `android.widget.LinearLayout` — горизонтальное упорядочивание виджетов. Пример изображен в Листинге 12.8.

Листинг 12.8. Исследования горизонтального
упорядочивания виджетов

```
<LinearLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
```

```
xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:text="Button1"
        android:layout_weight="1"
        />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button2"
        android:layout_weight="1"
        android:layout_gravity="bottom"
        />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:text="Button3"
        android:layout_weight="1"
        />
</LinearLayout>
```

Результат Листинга 12.8 смотрите на Рис. 12.5.

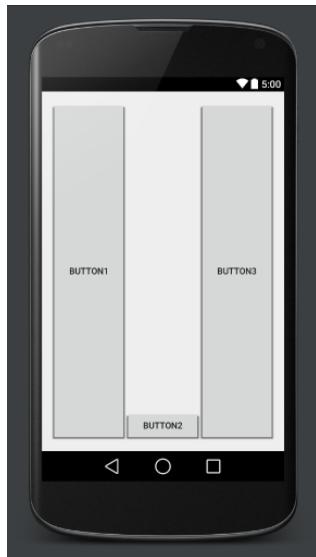


Рис. 12.5. Результат верстки из Листинга 12.8

12.3. TableLayout

Менеджер раскладки `android.widget.TableLayout` упорядочивает свои дочерние виджеты по строкам и столбцам.

Иерархия классов:

```
java.lang.Object
 |
 +-- android.view.View
   |
   +--- android.view.ViewGroup
     |
     +--- android.widget.LinearLayout
       |
       +--- android.widget.TableLayout
         |
         +--- android.widget.TableRow
```

Как видно из иерархии классов, класс менеджера раскладки **android.widget.TableLayout** является дочерним от класса **android.widget.LinearLayout**. Следовательно, какое-то сходство между этими классами существует и нам будет проще разобраться с этим менеджером раскладки.

Полное описание класса **android.widget.TableLayout** можно прочитать по адресу <http://developer.android.com/reference/android/widget/TableLayout.html>.

Менеджер раскладки **android.widget.TableLayout** должен состоять из строк, которые представлены менеджером раскладки **android.widget.TableRow** (<http://developer.android.com/reference/android/widget/TableRow.html>). Таким образом, менеджер **android.widget.TableLayout** это **android.widget.LinearLayout** с вертикальным упорядочиванием дочерних виджетов, которыми являются контейнеры **android.widget.TableRow**. Менеджер раскладки **android.widget.TableRow** представляет из себя тот же **android.widget.LinearLayout** но с горизонтальным упорядочиванием дочерних виджетов. Причем дочерним виджетам менеджера **android.widget.TableRow** не нужно указывать атрибуты **android:layout_width** и **android_layout_height** (эти атрибуты для него являются не обязательными).

Давайте посмотрим на xml верстку для **android.widget.TableLayout** в Листинге 12.9.

Листинг 12.9. xml структура менеджера
android.widget.TableLayout

```
<TableLayout  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">
```

```
<TableRow>
    <Button android:text="Button11" />
    <Button android:text="Button12" />
    <Button android:text="Button13" />
</TableRow>

<TableRow>
    <Button android:text="Button21" />
    <Button android:text="Button22" />
    <Button android:text="Button23" />
</TableRow>
</TableLayout>
```

Необходимо напомнить вам, что вся xml верстка для менеджеров раскладки осуществляется в xml файле макета Активности (имя_модуля/res/layout/activity_main.xml).

Как выглядит результат исполнения кода из Листинга 12.9 можно увидеть на Рис. 12.6.

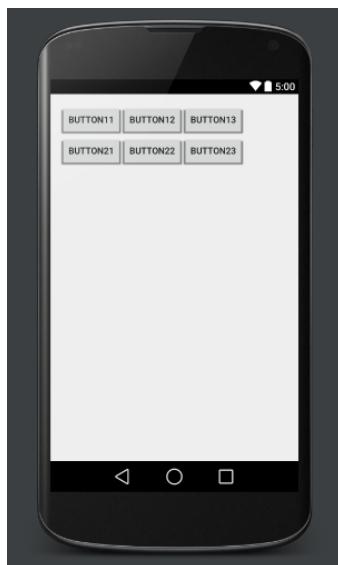


Рис. 12.6. Вид верстки макета Активности из Листинга 12.9

Как видно из Листинга 12.9. и изображения 12.6, добавление дочерних виджетов осуществляется по порядку. Но есть возможность управлять месторасположением конкретного виджета с помощью атрибута **android:layout_column**, которому присваивается номер столбца (нумерация от 0) в котором необходимо разместить виджет. Добавим еще одну строку **TableRow** в **android.widget.TableLayout** (см. Листинг 12.10) и назначим виджету с текстом «Button33» позицию в столбце 2.

Листинг 12.10. Позиционирование виджета
в конкретный столбец

```
<TableLayout>
    ...
    <TableRow>
        <Button android:text="Button31"/>

        <Button
            android:text="Button33"
            android:layout_column="2"
        />
    </TableRow>
</TableLayout>
```

Результат исполнения кода из Листинга 12.10 см. на Рис. 12.7.

Так же у менеджера **android.widget.TableLayout** есть возможность объединять ячейки с помощью атрибута **android:layout_span** которому присваивается числовое значение сколько объединить ячеек для размещения виджета. На Листинге 12.11 показано как использовать данный атрибут. Увидеть результат исполнения Листинга 12.11 можно на Рис. 12.8.



Рис. 12.7. Позиционирование виджета в конкретный столбец



Рис. 12.8. Объединение ячеек из Листинга 12.11

Листинг 12.11. Объединение ячеек

```
<TableLayout>
    ...
    <TableRow>
        <Button android:text="Button41" />
        <Button android:text="Button42"
            android:layout_span="2" />
    </TableRow>
</TableLayout>
```

И конечно же есть возможность по растягиванию ячеек по ширине на всю оставшуюся область родительского контейнера. За эту возможность отвечает атрибут **android:stretchColumns**, которому в качестве значения передается номер столбца, ячейки в котором должны растянуться таким образом, чтобы таблица заняла по ширине всю родительскую область контейнера. Нумерация ячеек от 0. Так же в качестве значения для этого атрибута можно передать «*****» — это означает, что должны растянуться ячейки всех столбцов. В Листинге 12.12 показано использование этого атрибута.

Листинг 12.12. Растворение ячеек

```
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="*">
    ...
</TableLayout>
```

Результат исполнения листинга 12.12 (с разными значениями для атрибута **android:stretchColumns**: '0', '1', '*****') можно увидеть на Рис. 12.9.

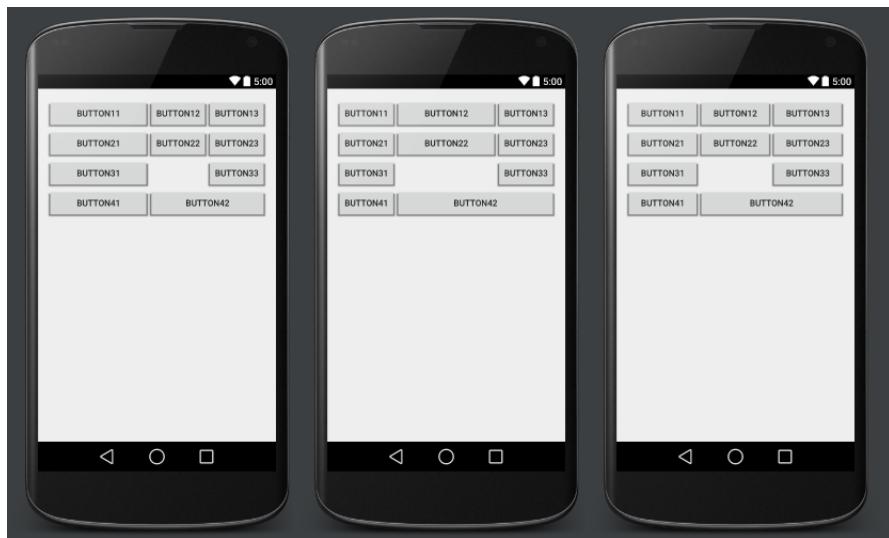


Рис. 12.9. Результат работы кода Листинга 12.12
с разными значениями атрибута

13. Программное создание элементов управления Button, TextView, EditText

Верстать макеты можно не только с помощью xml. Возможно и программное создание виджетов и размещение их на Активности. Этому и посвящена эта глава урока. Для примера программного создания виджетов создан модуль «app4» в исходных кодах, прилагаемых к уроку.

Правила программного создания виджетов не сложные. Во-первых, нужно создать объект виджета с помощью оператора **new**, как вы это могли делать ранее на других предметах по программированию. Конструкторы классов виджетов Android обязательно принимают в качестве параметра ссылку на объект **android.content.Context**. После создания виджета можно выполнить его настройку (разместить в нем текст, задать размеры, стили), после чего виджет добавляется в родительский контейнер с помощью метода **addView(View view)** родительского контейнера.

В Листинге 13.1 показано создание виджетов **EditText**, **Button**, **TextView**. Пример демонстрирует следующую функциональность. Виджеты создаются в обработчике события **onCreate** Активности. На кнопку назначается обработчик события клика, в котором происходит считывание текста, который ввел пользователь, из виджета **EditText** и размещение этого текста в виджете **TextView**. Функциональность не сложная. Для удобства реализации

этой функциональности, ссылки на **EditText** и **TextView** сделаны полями класса Активности — так как доступ к этим ссылкам необходим из разных методов класса Активности. Вся реализация примера находится в модуле «app4» исходного кода, прилагаемого к данному уроку.

Листинг 13.1. Программное создание виджетов **TextView**, **Button**, **EditText** и применение их в функционале примера

```
public class MainActivity extends ActionBarActivity
{
    //--- Class members -----
    /**
     * Ссылка на динамически создаваемый виджет EditText
     */
    private EditText edt1;

    /**
     * Ссылка на динамически создаваемый виджет TextView
     */
    private TextView tv1;

    //--- Class methods -----
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //--- Получение ссылки на главный контейнер Активности -
        LinearLayout LL = (LinearLayout) this.
            findViewById(R.id.mainLL);

        //--- Создание виджета EditText и добавление его
        //--- в контейнер -----
        this.edt1 = new EditText(this);
        this.edt1.setText("Type the text");
        LL.addView(this.edt1);
    }
}
```

```
//-- Создание виджета Button и добавление его
//-- в контейнер -----
    Button B = new Button(this);
    B.setText("Button");
    LL.addView(B);

//-- Назначение обработчика события клика на кнопку -
    B.setOnClickListener(new View.OnClickListener()
    {
        @Override
        public void onClick(View v)
        {

//-- Получение текста из виджета EditText -----
        String txt = MainActivity.this.edt1.
                    getText().toString();

//-- Размещение текста в виджете TextView -----
        MainActivity.this.tvl.setText(txt);
    }
});

//-- Создание виджета TextView и добавление его
//-- в контейнер -----
    this.tvl = new TextView(this);
    this.tvl.setText("Hello World");
    LL.addView(this.tvl);
}

...
}
```

14. Программное создание LinearLayout, TableLayout

Поскольку менеджеры раскладки так же являются виджетами, то их можно создавать программно аналогично программному созданию обычных виджетов. Для примера создания менеджеров компоновки создан модуль «app5», который можно найти в файлах с исходными кодами, прилагаемым к этому уроку. Пример создания менеджеров раскладки и наполнение их дочерними виджетами можно увидеть в Листинге 14.1.

Листинг 14.1. Программное создание менеджеров LinearLayout и TableLayout

```
public class MainActivity extends ActionBarActivity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //--- Получение ссылки на главный контейнер
        //--- Активности -----
        LinearLayout mainLL = (LinearLayout)
            this.findViewById(R.id.mainLL);

        //--- Создание менеджера LinearLayout
        //--- с вертикальным упорядочиванием -----
        LinearLayout LL = new LinearLayout(this);
```

```
LL.setOrientation(LinearLayout.VERTICAL);
mainLL.addView(LL);

//-- Прикрепление нескольких кнопок к LinearLayout
for (int i = 0; i < 3; i++)
{
    Button      B      = new Button(this);
    B.setText("Button" + (i + 1));
    LL.addView(B);
}

//-- Создание менеджера TableLayout с двумя строками
TableLayout   TL   = new TableLayout(this);
TL.setStretchAllColumns(true);
mainLL.addView(TL);

//-- Добавление первой строки в TableLayout -----
TableRow      TR1   = new TableRow(this);
TL.addView(TR1);

//-- Добавление нескольких кнопок в первую строку --
for (int i = 0; i < 3; i++)
{
    Button      B      = new Button(this);
    B.setText("Button1" + (i + 1));
    TR1.addView(B);
}

//-- Добавление второй строки в TableLayout -----
TableRow      TR2   = new TableRow(this);
TL.addView(TR2);
//-- Добавление нескольких кнопок в первую строку --
for (int i = 0; i < 3; i++)
{
    Button      B      = new Button(this);
```

```
B.setText("Button2" + (i + 1));  
TR2.addView(B);  
  
}  
}  
...  
}
```



Рис. 14.1. Результат исполнения примера Листинга 14.1

Результат исполнения кода примера из Листинга 14.1 показан на Рис. 14.1.

15. Программная «верстка» макетов Активности

В предыдущих двух примерах и главах программное создание виджетов и менеджеров раскладки осуществлялось с последующим их добавлением в главный контейнер Активности. В этой главе мы рассмотрим полное программное создание макета Активности и назначение созданного макета Активности. В принципе, механизм ничем не отличается от механизма описанного в предыдущих двух главах. Разница лишь в том, что главный контейнер Активности так же создается программно и сам макет назначается Активности с помощью вызова метода

```
public void setContentView (View view);
```

класса Активности. В качестве параметра, этот метод принимает виджет, который является главным виджетом (чаще всего — контейнером) Активности.

Для примера создан модуль «аррб», полный исходный код которого можно найти в файлах исходного кода, прилагаемых к этому уроку.

В примере будет программно «сверстан» макет Активности и с помощью меню пользователь сможет назначать Активности разные макеты (сверстанный с помощью xml и созданный программно).

Первым делом добавим строковые константы в файл ресурсов **имя_модуля/res/values/strings.xml** (Листинг 15.1).

Листинг 15.1. Файл ресурсов /res/values/strings.xml

```
<resources>
    <string name="app_name">MyApplication</string>

    <string name="action_xml">Назначить XML макет
                                    </string>
    <string name="action_prog">Назначить программный
                                макет</string>
    <string name="action_settings">Settings</string>
</resources>
```

Как вы догадались, созданные строковые константы предназначены для пунктов меню. Следующий шаг, добавим пункты меню в файл **имя_модуля/res/menu/menu_main.xml** (Листинг 15.2).

Листинг 15.2. Добавление пунктов меню в пример приложения

```
<menu xmlns:android=
      "http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      tools:context=".MainActivity">

    <item android:id="@+id/action_settings"
          android:title="@string/action_settings"
          android:orderInCategory="100"
          app:showAsAction="never"
          />

    <item android:id="@+id/action_xml"
          android:title="@string/action_xml"
          android:orderInCategory="110"
          app:showAsAction="never"
          />
```

```

<item android:id="@+id/action_prog"
      android:title="@string/action_prog"
      android:orderInCategory="120"
      app:showAsAction="never"
/>

```

Следующий шаг — сверстаем несложный XML макет Активности (Листинг 15.3).

Листинг 15.3. XML макет Активности для примера приложения

```

<LinearLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"
    android:layout_height="match_parent"

    tools:context=".MainActivity"
    android:orientation="vertical"
    >

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/btn1"
        android:text="1" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/btn2"
        android:text="2" />

```

```
<Button  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:id="@+id(btn3)"  
    android:text="3" />  
  
<Button  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:id="@+id	btn4"  
    android:text="4" />  
  
<Button  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:id="@+id	btn5"  
    android:text="5" />  
  
<Button  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:id="@+id	btn6"  
    android:text="6" />  
  
</LinearLayout>
```

В макете сверстаны шесть кнопок, располагаемые друг под другом. Этот пример макета сделан с заделом на будущее — он пригодится нам в следующей главе (Примечание: Надписи на кнопках в этом примере носят чисто технический характер, поэтому для них не создавались строковые константы).

Следующий шаг — программная верстка макета в обработчике **onCreate** (Листинг 15.4).

Листинг 15.4.Программная «верстка» макета и назначение его Активности

```

public class MainActivity extends ActionBarActivity

{
    //--- Class members -----
    /**
     * Ссылка на макет активности, созданный программно
     */
    private TableLayout progModel;

    //--- Class methods -----
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);
        //Закомментировано !!

        //--- Создание макета для Активности программно ---
        this.progModel = new TableLayout(this);
        this.progModel.setStretchAllColumns(true);

        for (int i = 0; i < 4; i++)
        {
            TableRow TR = new TableRow(this);
            this.progModel.addView(TR);

            for (int j = 0; j < 4; j++)
            {
                Button B = new Button(this);
                B.setText(String.valueOf((i + 1) * 10 +
                    (j + 1)));
                TR.addView(B);
            }
        }
    }
}

```

```

    //--- Назначение созданного макета Активности -----
        this.setContentView(this.progModel);
    }
    ...
}

```

Обратите внимание, как созданный программно макет назначается Активности (выделено в Листинге 15.4 жирным).

Последний шаг — обработка событий от пунктов меню и поочередная смена макетов Активности (Листинг 15.5).

Листинг 15.5. Переключение между макетами для Активности

```

@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    int id = item.getItemId();

    switch (id)
    {
        case R.id.action_settings :
            return      true;

//--- Назначение Активности xml макета из ресурсов --
        case R.id.action_xml :
            this.setContentView(R.layout.activity_main);
            return      true;

//--- Назначение Активности макета, созданного
//--- программно -----
        case R.id.action_prog :
            this.setContentView(this.progModel);
            return      true;
    }
    return super.onOptionsItemSelected(item);
}

```

В Листинге 15.5 жирным выделен код назначения макета Активности.

На Рис. 15.1 можно увидеть внешний вид приложения с разными макетами Активности (снимок слева — программный макет, справа — xml макет).



Рис. 15.1. Слева — программно созданный макет, справа — xml макет из ресурсов

16. Поворот экрана. Создание макета Активности для альбомной ориентации

Устройство можно поворачивать. Как вы уже знаете, при повороте устройства объект Активности создается заново. И этот факт нужно учитывать при разработке Android приложений. Но не только этот. Может так случится, что макет Активности, сверстанный разработчиком для вертикальной (портретной) ориентации экрана (ориентация по умолчанию при создании модуля) может не отобразится корректно при горизонтальной (альбомной) ориентации. В таком случае, необходим дополнительный макет для Активности, который должен быть назначен Активности при горизонтальной ориентации. В этой главе мы научимся это делать.

Для примера будем использовать xml макет из предыдущей главы (Листинг 15.3), но в новом модуле (модуль «app7»). Попробуйте повернуть устройство в эмуляторе (в Google эмуляторе нужно нажать комбинацию клавиш Ctrl+F11, в Genymotion нажать на значок поворота в панели инструментов справа).

Внешний вид макета Активности в устройстве при вертикальной ориентации изображен на Рис. 16.1.

Для добавления макета Активности горизонтальной (альбомной) ориентации кликнем правой кнопкой мыши по папке «layout» модуля проекта в Android Studio.



Рис. 16.1. Макет Активности для вертикальной (портретной) ориентации

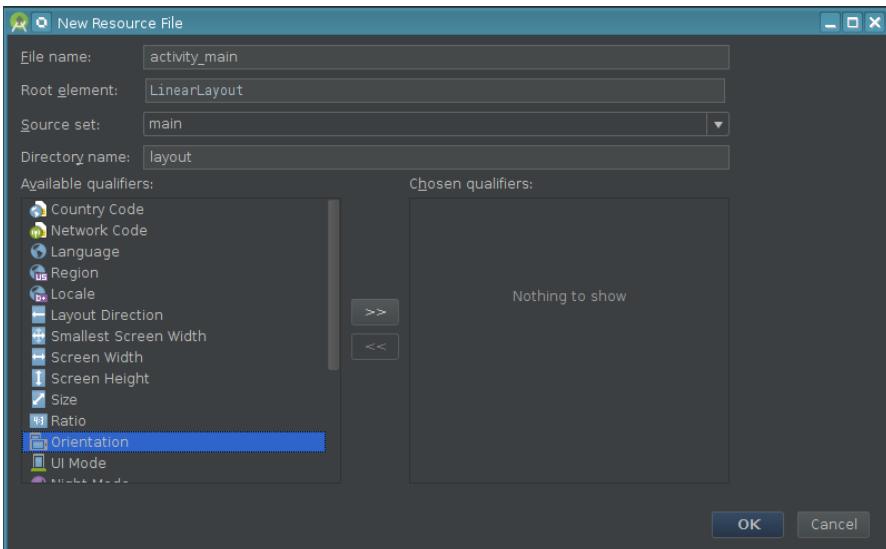


Рис. 16.2. Диалоговое окно добавления нового файла ресурса Активности

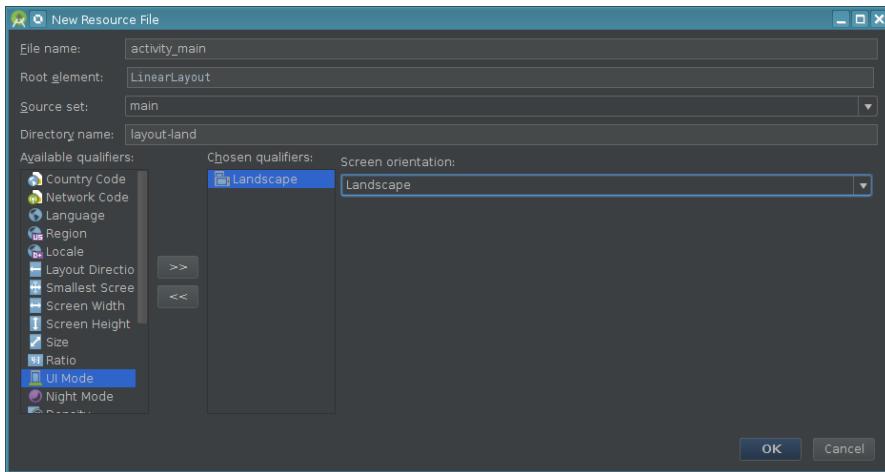


Рис. 16.3. Выбор ориентации для нового файла макета Активности

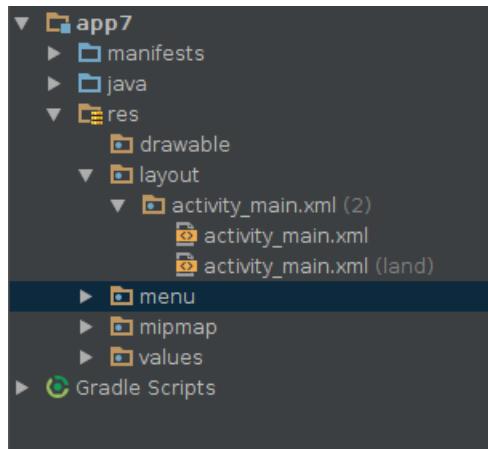


Рис. 16.4. Файл макета Активности альбомной ориентации

Появится контекстное меню, в котором выберем опцию «New». Появится следующее выпадающее контекстное меню, в котором выберем опцию «Layout File».

В появившемся окне (Рис. 16.2) введите имя файла в опцию «File Name». Далее выберите опцию «Orientation» и нажмите на кнопку «>>>» (Рис. 16.3). В выпадающем списке выберите ориентацию «Landscape» (альбомная). Нажимайте кнопку «OK».

В модуле нашего проекта в папке «layout» появится созданный нами файл ресурсов (см. Рис. 16.4) с таким же названием, но с примечанием «(land)».

Редактируем этот файл — создаем там макет Активности для альбомной ориентации (Листинг 16.1).

Листинг 16.1. Верстка макета Активности для альбомной ориентации

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

<TableLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="*>

    <TableRow>
        <Button android:id="@+id	btn1" android:text="1" />
        <Button android:id="@+id	btn2" android:text="2" />
        <Button android:id="@+id	btn3" android:text="3" />
    </TableRow>

    <TableRow>
        <Button android:id="@+id	btn4" android:text="4" />
        <Button android:id="@+id	btn5" android:text="5" />
        <Button android:id="@+id	btn6" android:text="6" />
    </TableRow>

</TableLayout>
```

Все. Запускаем наше приложение в эмуляторе. Поворачиваем устройство в горизонтальную ориентацию и наблюдаем смену макета на альбомную ориентацию (Рис. 16.5).

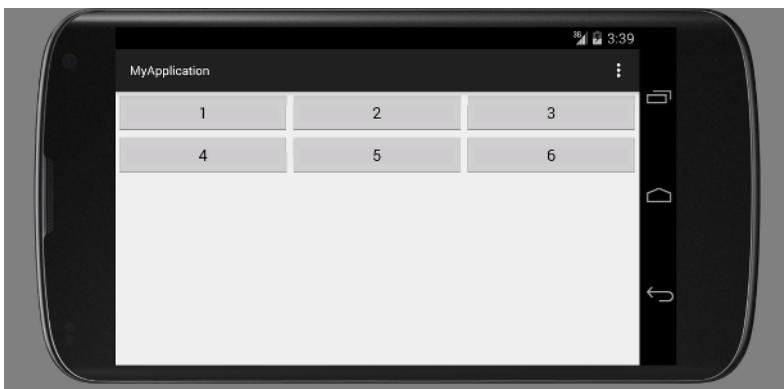
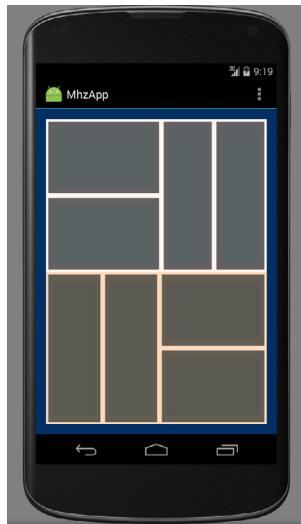


Рис. 16.5. Альбомная ориентация с макетом из Листинга 16.1

17. Домашнее задание

- Сверстайте макет активности, как показано на Рисунке ниже.



- Напишите приложение, в котором пользовательводит указателем по экрану устройства, и если движение близко по траектории к вертикальному, то цвет фона Активности становится желтым, если движение указателя пользователя близко по траектории к горизонтальному, то цвет фона Активности становится зеленым. И наконец, если движение диагональное, то цвет Активности становится красным. Впрочем, цвета вы можете использовать другие — на ваш вкус 😊

Для изменения цвета фона Активности используйте вызов метода:

```
public void setBackgroundColor(int Color);
```

для главного контейнера Активности. Подробнее об этом методе прочтите в API.

3. Создайте приложение "Калькулятор" (ну, куда же без него 😊) выполняющий операции сложения, вычитания, умножения, деления над целыми числами. Внешний вид калькулятора должен быть похож на обычный оконный кнопочный калькулятор.



Урок №1

Введение в Android. Основы Android.

Установка необходимого ПО

© Бояршинов Юрий

© Компьютерная Академия «Шаг»

www.itstep.org

Все права на охраняемые авторским правом фото-, аудио- и видеопроизведения, фрагменты которых использованы в материале, принадлежат их законным владельцам. Фрагменты произведений используются в иллюстративных целях в объеме, оправданном поставленной задачей, в рамках учебного процесса и в учебных целях, в соответствии со ст. 1274 ч. 4 ГК РФ и ст. 21 и 23 Закона Украины «Про авторське право і суміжні права». Объем и способ цитируемых произведений соответствует принятым нормам, не наносит ущерба нормальному использованию объектов авторского права и не ущемляет законные интересы автора и правообладателей. Цитируемые фрагменты произведений на момент использования не могут быть заменены альтернативными, не охраняемыми авторским правом аналогами, и как таковые соответствуют критериям добросовестного использования и честного использования.

Все права защищены. Полное или частичное копирование материалов запрещено. Согласование использования произведений или их фрагментов производится с авторами и правообладателями. Согласованное использование материалов возможно только при указании источника.

Ответственность за несанкционированное копирование и коммерческое использование материалов определяется действующим законодательством Украины.