Министерство образования и науки Российской Федерации

Калужский филиал
федерального государственного бюджетного образовательного
учреждения высшего образования
**«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»**
(КФ МГТУ им. Н.Э. Баумана)

Ю.С. Белов, С.С. Гришунов

РАЗРАБОТКА МОБИЛЬНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Конспект лекций

Калуга – 2018

УДК 004.42
ББК 32.972.13
Б435

Конспект лекций составлен в соответствии с учебным планом КФ МГТУ им. Н.Э. Баумана по направлению подготовки 09.03.04 «Программная инженерия» кафедры «Программного обеспечения ЭВМ, информационных технологий и прикладной математики».

Конспект лекций рассмотрен и одобрен:

- Кафедрой «Программного обеспечения ЭВМ, информационных технологий и прикладной математики» (ФН1-КФ) протокол № 7 от « 21 » февраля 2018 г.

И.о. зав. кафедрой ФН1-КФ _____ к.т.н., доцент Ю.Е. Гагарин

- Методической комиссией факультета ФНК протокол № 2 от « 28 » 02 2018 г.

Председатель методической
комиссии факультета ФНК _____ к.х.н., доцент К.Л. Анфилов

- Методической комиссией
КФ МГТУ им.Н.Э. Баумана протокол № 2 от « 06 » 03 2018 г.

Председатель методической комиссии
КФ МГТУ им.Н.Э. Баумана
_____ д.э.н., профессор О.Л. Перерва

Рецензент:
к.ф.-м.н., директор по исследованиям
и развитию ООО "НПФ "Эверест" _____ В.Ю. Кириллов

Авторы
к.ф.-м.н., доцент кафедры ФН1-КФ _____ Ю.С. Белов
асс. кафедры ФН1-КФ _____ С.С. Гришунов

Аннотация

Конспект лекций включает сведения о платформе Android и ее особеностях: основных элементах управления, типов меню, различных способах хранения данных, работе с мультимедиа и графикой, создания виджетов и служб.

Предназначены для студентов 4-го курса бакалавриата КФ МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Настоящий курс лекций составлен в соответствии с программой лекционных занятий по курсу «Разработка мобильного программного обеспечения» на кафедре «Программное обеспечение ЭВМ, информационные технологии и прикладная математика» факультета фундаментальных наук Калужского филиала МГТУ им. Н.Э. Баумана.

Курс лекций, ориентирован на студентов 3-го курса направления подготовки 09.03.04 «Программная инженерия», содержит сведения о платформе Android и ее особеностях: основных элементах управления, типов меню, создании многоэкранных приложений, различных способах хранения данных, а также создания виджетов и служб.

Модуль 1. СОЗДАНИЕ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ НА ОСНОВЕ ПРОСТЫХ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ

Лекция 1.1. ВВЕДЕНИЕ В ANDROID. ПЕРВОЕ ANDROID ПРИЛОЖЕНИЕ. КОМПОНОВКА ЭЛЕМЕНТОВ УПРАВЛЕНИЯ

Вот как описывает Android Энди Рубин из Google:

Первая действительно открытая и всеобъемлющая платформа для мобильных устройств и любого программного обеспечения, предназначенного для работы на мобильном телефоне, при этом без патентных ограничений, которые сдерживали развитие портативных устройств.

(_ http://googleblog.blogspot.com/2007/11/wheres-my-gphone.html _).

Упрощенно Android можно представить, как комбинацию трех компонентов:

- свободной операционной системы с открытыми исходными кодами;
- среды разработки с открытыми исходными кодами для создания мобильных приложений;
- устройств, по большей части мобильных телефонов, на которых установлена операционная система Android вместе с разработанными для нее приложениями.

Главным преимуществом Android как среды разработки стал ее API.

Android как нейтральная к приложениям платформа предоставляет возможность создавать программы, которые станут такой же неотъемлемой частью телефона, как и компоненты, поставляемые в комплекте.

Следующий список иллюстрирует основные характеристики Android:

- отсутствие расходов на использование лицензии, распространение и разработку, а также каких-либо механизмов сертификации готовых программных продуктов;
- доступ к Wi-Fi-устройству;
- в сетях GSM, EDGE и 3G, предназначенных для телефонии и передачи данных, можно звонить или принимать звонки и SMS, отправлять и получать данные;
- комплексный API для работы с навигационными службами, например, GPS;
- полный контроль над мультимедийными устройствами, включая проигрывание или запись информации с камеры и микрофона;
- API для работы с сенсорными устройствами, например, акселерометром и компасом;
- библиотеки для работы с Bluetooth с возможностью передачи данных по протоколу p2p;
- хранилища для общих данных;
- фоновые приложения и процессы;
- виджеты для Рабочего стола, Живые каталоги (Live Folders) и живые обои (Live Wallpaper);
- возможность интеграции результатов поиска приложения в системный поиск;
- встроенный браузер на базе WebKit с открытыми исходными кодами и поддержкой HTML5;
- полная поддержка приложений, которые используют функционал работы с картами в своем пользовательском интерфейсе;
- оптимизированная под мобильные устройства графическая система с аппаратным ускорением, включающая библиотеку для работы с векторной 2D-графикой и поддержку трехмерной графики с использованием OpenGL ES 2.0;

– мультимедийные библиотеки для проигрывания и записи аудио-, видеофайлов или изображений;
– локализация с помощью инструментов для работы с динамическими ресурсами;
– набор программных компонентов для повторного использования компонентов и замещения встроенных приложений.

Open Handset Alliance (OHA) — бизнес-альянс 48 компаний по разработке открытых стандартов для мобильных устройств, включающий Google, HTC, Intel, Motorola, Qualcomm, Samsung, LG, T-Mobile, Nvidia, Wind River Systems и другие компании.

Альянс открытых мобильных устройств — это сообщество из более чем 50 компаний, включающее производителей аппаратного и программного обеспечения, а также мобильных операторов. Среди наиболее значительных членов Альянса можно назвать компании Motorola, HTC, T-Mobile и Qualcomm. Вот как формулируют основные идеи OHA участники этого сообщества:

«Приверженность открытости, общее видение будущего и конкретные задачи для воплощения мечты в реальность. Ускорение внедрения инноваций в сфере мобильных технологий и предоставление потребителям функциональных, менее дорогих и более продвинутых мобильных устройств».

OHA была основана 5 ноября 2007 года под предводительством Google и 34 прочих членов, включающих производителей мобильных телефонов, разработчиков программного обеспечения, некоторых мобильных поставщиков и изготовителей чипов. Nokia, AT&T и Verizon Wireless не являются членами альянса, однако Verizon недавно выразил желание использовать Android в будущем, указывая на возможность скорого вступления в альянс. Android, основной программный пакет альянса, основан на открытом исходном коде и будет конкурировать с другими мобильными платформами от Apple Inc., Microsoft, Nokia, Palm, Research In Motion и Symbian.

Одновременно с объявлением о формировании Open Handset Alliance 5 ноября 2007, OHA также обнародовала информацию об Android, открытой платформе для мобильных телефонов, основанной на ядре Linux. Первая версия SDK для разработчиков была выпущена 12 ноября 2007.

Первым коммерческим телефон, использующим Android, является T-Mobile G1 (также известный как HTC Dream). Он был одобрен FCC 18 августа 2008 и стал доступен 22 октября. Состояние рынка мобильных платформ представлено на рисунках 1.1 – 1.4.

Top Five Smartphone Operating Systems, Shipments, and Market Share, 2013 (Units in Millions)

Operating System	2013 Shipment Volumes	2013 Market Share	2012 Shipment Volumes	2012 Market Share	Year-Over-Year Change
Android	793.6	78.6%	500.1	69.0%	58.7%
iOS	153.4	15.2%	135.9	18.7%	12.9%
Windows Phone	33.4	3.3%	17.5	2.4%	90.9%
BlackBerry	19.2	1.9%	32.5	4.5%	-40.9%
Others	10.0	1.0%	39.3	5.4%	-74.6%
Total	**1009.6**	**100.0%**	**725.3**	**100.0%**	**39.2%**

Source: IDC Worldwide Mobile Phone Tracker, February 12, 2014

Рис. 1.1

Top Five Smartphone Operating Systems, Shipments, and Market Share, 4Q 2013 (Units in Millions)

Operating System	4Q13 Shipment Volumes	4Q13 Market Share	4Q12 Shipment Volumes	4Q12 Market Share	Year-Over-Year Change
Android	226.1	78.1%	161.1	70.3%	40.3%
iOS	51.0	17.6%	47.8	20.9%	6.7%
Windows Phone	8.8	3.0%	6.0	2.6%	46.7%
BlackBerry	1.7	0.6%	7.4	3.2%	-77.0%
Others	2.0	0.7%	6.7	2.9%	-70.1%
Total	**289.6**	**100.0%**	**229.0**	**100.0%**	**26.5%**

Source: IDC Worldwide Mobile Phone Tracker, February 12, 2014

Рис. 1.2

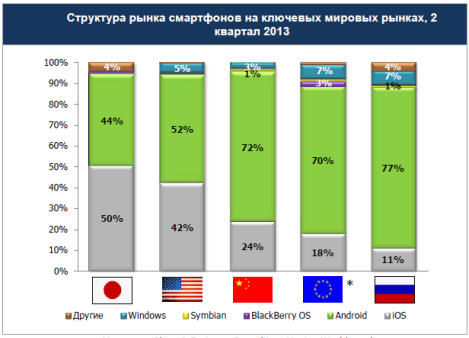

Рис. 1.3

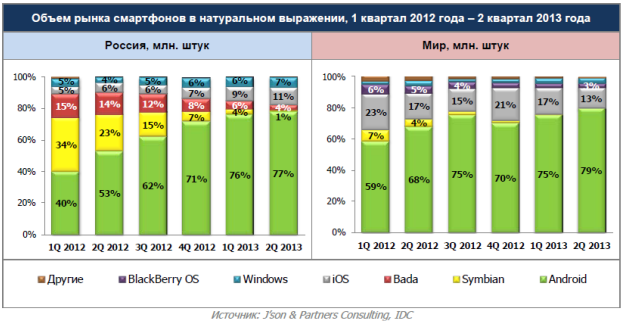

Рис. 1.4

Архитектура Android состоит из четырех уровней: уровень ядра, уровень библиотек и среды выполнения, уровень каркаса приложений (application framework) и уровень приложений.

Система Android основана на ядре Linux версии 2.6. Тем не менее Android не является Linux-системой в прямом смысле этого слова. У Android свои механизмы распределения памяти, другая система межпроцессного взаимодействия (Inter-Process Communication, IPC), специфические модули ядра и т. д. На уровне ядра происходит управление аппаратными средствами мобильного устройства. На этом уровне работают драйверы дисплея, камеры, клавиатуры, WiFi, аудиодрайверы. Особое место занимают драйверы управления

питанием и драйвер межпроцессного взаимодействия (IPC). Linux обеспечивает уровень абстракций между оборудованием и остальными частями стека Android. С точки зрения внутренней архитектуры Android использует Linux для управления памятью, процессами, сетевым взаимодействием и другими возможностями операционной системы.

На этом уровне также расположен набор драйверов для обеспечения работы с оборудованием мобильного устройства. Набор драйверов может отличаться в зависимости от производителя и модели устройства. Поскольку новое вспомогательное оборудование для мобильных устройств постоянно появляется на рынке, драйверы для них должны быть написаны на уровне ядра Linux для обеспечения поддержки оборудования, также как и для настольных Linux-систем.

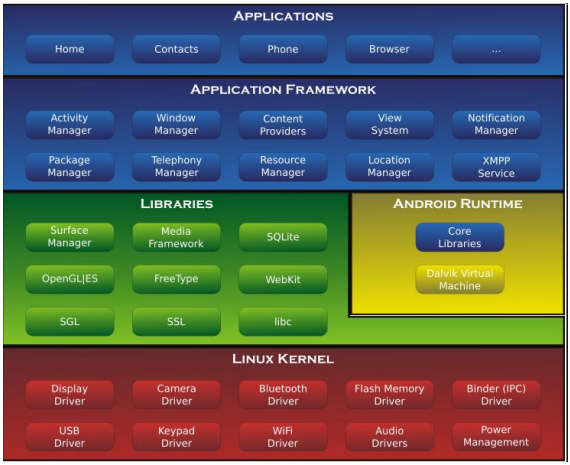

Рис. 1.5

Следующий уровень над ядром Linux включает набор библиотек C/C++, используемых различными компонентами ОС. Библиотеки этого уровня по своему функциональному назначению можно разделить на две группы:

– системная библиотека C;
– функциональные библиотеки C/C++.

Системная библиотека базируется на Berkeley Software Distribution (BSD). Компания Google разработала собственную версию системной библиотеки libc — Bionic специально для мобильных устройств на основе Linux. Это было необходимо для обеспечения быстрой загрузки библиотеки в каждый процесс, и, следовательно, библиотека должна была иметь маленький размер. Библиотека Bionic имеет размер около 200 Кбайт, что в два раза меньше размера стандартной библиотеки Linux glibc. Кроме того, необходимо было учитывать ограниченную мощность центрального процессора мобильного устройства. Это означает, что библиотека должна быть оптимизирована для максимального быстродействия. Конечно, сейчас это уже не актуально, современные мобильные устройства практически сравнялись по мощности процессора с нетбуками, но еще несколько лет назад это являлось серьезной проблемой.

Библиотека Bionic имеет встроенную поддержку важных для Android системных служб и регистрацию системных событий, но в то же время она не поддерживает некоторую функциональность, например, исключения C++, и несовместима с GNU libc и стандартом POSIX.

Уровень, следующий за уровнем ядра, содержит исходные библиотеки Android. Эти разделяемые библиотеки написаны на C или на C++, скомпилированы для конкретной аппаратной архитектуры и предустановлены на устройство разработчиком телефона.

Рассмотрим некоторые наиболее важные исходные библиотеки:

Менеджер поверхностей. Android использует композитный менеджер окон, похожий на Vista или Compiz, но более простой. Вместо того чтобы выводить графические данные непосредственно в буфер экрана, команды отображения графики формируют закадровые битовые массивы, которые затем объединяются с другими массивами

для того, чтобы сформировать изображение, которое видит пользователь. Это позволяет системе создавать различные интересные эффекты, например, полупрозрачные окна и градиентные переходы.

2D и 3D графика. В Android двух- и трехмерные графические элементы комбинируются в единый пользовательский интерфейс. Библиотека будет использовать возможности аппаратного 3D-ускорения, если устройство ими оснащено, или быстрый программный рендеринг, если нет.

Медиа-кодеки. Android может проигрывать видеоролики и фильмы, записывать и воспроизводить аудио фрагменты в различных форматах, в том числе AAC, AVC (H.264), H.263, MP3 и MPEG4.

База данных SQL. Android имеет «легковесную» встраиваемую реляционную базу данных SQLite, эта же база данных используется в Firefox и в Apple iPhone. Используйте этот механизм для постоянного хранения данных ваших приложений.

Браузер. Для быстрого отображения HTML-контента Android использует библиотеку WebKit. Тот же механизм используется в браузере Google Chrome, браузере Apple Safari, в Apple iPhone и платформе Nokia S60.

Эти библиотеки не являются отдельными приложениями. Они существуют только для того, чтобы их могли вызывать высокоуровневые программы. Начиная с версии 1.5 Android позволяет писать и внедрять свои собственные библиотеки, используя Native Development Toolkit (NTK).

Среда выполнения Android также находится над ядром и включает в себя виртуальную машину Dalvik и библиотеки ядра Java.

Виртуальная машина (VM) Dalvik — это виртуальная машина Java в исполнении Google, оптимизированная для мобильных устройств. Весь код, который вы создаете для Android, пишется на Java и выполняется внутри виртуальной машины. Dalvik имеет следующие отличия от обычной Java-машины:

Dalvik VM запускает файлы .dex, которые конвертируются при компиляции из стандартных файлов .class и .jar. Файлы. dex более компактны и эффективны, чем файлы классов, что является важным соображением, если принять во внимание ограничения памяти и энергопотребления устройств, для которых предназначен Android.

Библиотеки ядра Java, которые поставляются с Android, отличаются и от библиотек Java Standard Edition (Java SE), и от библиотек Java Mobile Edition (Java ME). Однако они очень похожи.

Dalvik Virtual Machine — основанная на регистрах виртуальная машина, разработанная и написанная Дэном Борнштейном (англ. Dan Bornstein) и другими, как часть мобильной платформы Android.

Dalvik оптимизирован для низкого потребления памяти, это нестандартная регистр-ориентированная виртуальная машина, хорошо подходящая для исполнения на RISC-архитектурах процессоров, часто используемых в мобильных и встраиваемых устройствах, таких, как коммуникаторы и планшетные компьютеры. Большинство виртуальных машин, используемых на десктопах, являются стек-ориентированными, включая стандартную виртуальную машину Java от Oracle.

Программы для Dalvik пишутся на языке Java. Несмотря на это, стандартный байт-код Java не используется, вместо него Dalvik VM исполняет байткод собственного формата. После компиляции исходных текстов программы на Java (при помощи javac) утилита dx из «Android SDK» преобразует. class файлы в формат. dex, пригодный для интерпретации в Dalvik.

Следующий уровень — уровень каркаса приложений. Уровень каркаса приложений находится над библиотеками и средой выполнения.

Он обеспечивает высокоуровневые строительные блоки, которые вы будете использовать для создания приложений. Фреймворк поставляется предустановленным вместе с Android, но при

необходимости вы можете расширять его с помощью собственных компонентов. На этом уровне работают различные диспетчеры:

- Диспетчер активности (Activity Manager) — управляет жизненным циклом приложения;
- Диспетчер пакетов (Package Manager) — управляет установкой пакетов прикладных программ;
- Диспетчер окон (Window Manager) — управляет окнами приложений;
- Диспетчер ресурсов (Resource Manager) — используется для доступа к строковым, графическим и другим типам ресурсов;
- Контент-провайдеры (Content Providers) — службы, предоставляющие приложениям доступ к данным других приложений;
- Диспетчер телефонии (Telephony Manager) — предоставляет API, с помощью которого можно контролировать основную телефонную информацию — статус подключения, тип сети и т. д.;
- Диспетчер местоположения (Location Manager) — позволяет приложениям получать информацию о текущем местоположении устройства;
- Диспетчер уведомлений (Notification Manager) — позволяет приложению отображать уведомления в строке состояния;

Система представлений (View System) — используется для создания внешнего вида приложения (позволяет организовать кнопки, списки, таблицы, поля ввода и другие элементы пользовательского интерфейса).

На уровне приложений работает большинство Android-приложений: браузер, календарь, почтовый клиент, навигационные карты и т. д. Нужно отметить, что Android не делает разницы между приложениями телефона и сторонними программами, поэтому любую стандартную

программу можно заменить альтернативной. При разработке приложений программист имеет полный доступ ко всем функциям операционной системы, что позволяет полностью переделать систему под себя.

Среда разработки приложений для Android включает все необходимое для создания, тестирования и отладки программ.

API-платформы android. Ядро среды разработки — библиотеки API, которые обеспечивают программисту доступ к стеку платформы Android. Эти же самые библиотеки используются компанией Google для написания встроенных в Android приложений.

Инструменты разработки. Вы можете преобразовать исходный код в исполняемые приложения для Android. В состав среды разработки входят инструменты, которые позволяют компилировать и отлаживать приложения.

Менеджер виртуальных устройств и эмулятор. Эмулятор Android — это полностью интерактивный эмулятор устройств, включающий несколько альтернативных вариантов интерфейса. Эмулятор запускается внутри виртуального устройства Android, которое моделирует аппаратную конфигурацию определенной модели телефона. С помощью эмулятора вы можете увидеть, как приложения будут выглядеть и функционировать на реальном устройстве под управлением Android. Все программы здесь запускаются внутри виртуальной машины Dalvik, программный эмулятор создает для них идеальное окружение, которое не зависит от особенностей той или иной аппаратной начинки и представляет собой лучшую независимую среду для тестирования, чем любая конкретная аппаратная реализация.

Полный набор документации. Среда разработки включает расширенную справочную информацию с примерами кода, в которой описывается, что входит в каждый программный пакет и класс и как их можно использовать. В дополнение к этому в справочной

документации содержится стартовый курс для начинающих, а также подробное описание основ разработки программ для Android.

Примеры кода. В среде разработки вы найдете примеры программ, которые демонстрируют некоторые возможности Android, а также несколько простых приложений, посвященных определенным функциям API.

Онлайн-поддержка. Группы Google по адресу http://developer.android.com/resources/community-groups.html — это форумы разработчиков, на которых часто появляются сообщения от команд инженеров и специалистов компании Google. Ресурс StackOverflow по адресу http://www.stackoverflow.com/questions/tagged/android также стал популярным — здесь публикуются вопросы, посвященные Android.

Требования к Android приложениям:

- **Приложение должно вести себя надлежащим образом**. Прежде всего убедитесь, что Активности приостанавливают работу, уходя в фоновый режим. Во время остановки или возобновления работы Активностей Android генерирует события, обрабатывая которые вы можете «заморозить» обновление графических элементов и отложить сетевые запросы, ведь если никто не видит ваш пользовательский интерфейс, нет никакого смысла его обновлять. Для случаев, когда работа должна продолжаться даже в фоне, в Android предусмотрен класс Service, не зависящий от графического интерфейса.
- **Приложение должно плавно переходить из фонового режима на передний план**. Учитывая многозадачность мобильных устройств, весьма вероятно, что ваши приложения будут регулярно уходить в фон и возвращаться обратно. Важно сделать так, чтобы они «возвращались к жизни» быстро и плавно. Управление процессами в Android недетерминированно: если ваше приложение находится в фоновом режиме, его работа

может преждевременно завершиться для освобождения ресурсов. Все это должно быть скрыто от пользователя. Вы можете обеспечивать цельность своего приложения, сохраняя его состояние и помещая обновления в очередь — пользователь будет думать, что работа программы просто была возобновлена, не замечая повторного запуска. Переключение между состояниями должно происходить плавно, а на экран нужно выводить тот же интерфейс, который был до этого.

- **Приложение должно быть деликатным**. Ваше приложение никогда не должно перехватывать ввод данных или прерывать работу текущей Активности. Если оно не на переднем плане, следует использовать объекты Notification и Toast, вместо того, чтобы привлекать внимание пользователя прямо из окна своей программы. Существует несколько способов, с помощью которых мобильное устройство может уведомлять пользователей о разных событиях. Например, при входящем звонке телефон проигрывает мелодию, при получении сообщения мигают светодиоды, а при обнаружении новой голосовой почты в статусной строке появляется значок в виде конверта. Эти и другие методики доступны благодаря механизму уведомлений.
- **Приложение должно иметь целостный и последовательный пользовательский интерфейс**. Ваше приложение, скорее всего, будет использоваться наряду с другими программами, поэтому важно, чтобы оно имело простой и понятный графический интерфейс. Не заставляйте пользователей приспосабливаться к приложению при каждом его запуске. Работа с ним должна быть легкой, простой и очевидной, особенно с учетом ограниченных размеров экрана и постоянных раздражителей, которые отвлекают от телефона.
- **Приложение должно быть отзывчивым**. Отзывчивость — один из наиболее важных факторов при проектировании

программ для мобильных устройств. Несомненно, вы уже успели испытать разочарование от приложений, которые «тормозят» во время работы, такие ситуации еще больше раздражают, если учитывать многофункциональную природу мобильных устройств. Рискуя столкнуться с задержками, вызванными медленными и ненадежными сетевыми подключениями, необходимо использовать потоки и фоновые Сервисы, чтобы ваши Активности не теряли отзывчивости. Что еще более важно — нужно предусмотреть остановку работы своих компонентов, чтобы другие приложения тоже могли выполняться как следует.

На обычном рабочем столе Linux или Windows вы запускаете множество приложений и просматриваете результаты их работы в отдельных окнах. Одно из рабочих окон «активировано», то есть владеет фокусом ввода, однако все программы равноправны между собой. По своему желанию вы можете переключаться между ними и перемещать их, для того чтобы видеть свои действия и закрывать программы, которые не нужны.

Android работает не так.

В Android есть приложение переднего плана, активное приложение, которое обычно занимает весь экран, кроме строки состояния. Когда пользователь включает свой телефон, первое приложение, которое он видит, - это программа Home (Домашний экран).

Когда пользователь запускает программу, Android начинает ее исполнение и делает ее активной. Из этого приложения пользователь может вызвать другое приложение или другой экран в том же самом приложении, и так далее. Все эти программы и экраны записываются в стек приложений (application stack) системным Менеджером деятельностей. В любое время пользователь может нажать кнопку Back и вернуться к предыдущему экрану в стеке. С точки зрения пользователя, это работает почти так же, как перемещение по истории

просмотров в веб-браузере, где нажатие кнопки Back возвращает его на предыдущую страницу.

Изнутри каждый экран пользовательского интерфейса представлен классом Activity. Каждая деятельность имеет собственный жизненный цикл. Приложение — это одна или несколько деятельностей плюс процесс Linux, содержащий их.

В отличие от большинства традиционных платформ в Android приложения имеют ограниченный контроль над жизненным циклом. Программные компоненты должны отслеживать изменения в состоянии приложения и реагировать на них соответствующим образом, уделяя особое внимание подготовке к преждевременному завершению работы.

По умолчанию каждое приложение в Android работает в собственном процессе — отдельном экземпляре виртуальной машины Dalvik. Управление памятью и процессами — исключительно прерогатива системы.

Android активно управляет своими ресурсами, делая все возможное, чтобы устройство оставалось отзывчивым. То есть работа процессов (вместе с приложениями, которые они в себе выполняют) в некоторых случаях может быть завершена без предупреждения. Это касается ситуаций, когда необходимо выделить ресурсы для приложений с более высоким приоритетом, которые, как правило, должны в этот момент взаимодействовать с пользователем.

Создавая объекты Activity, система помещает их в стек. При уничтожении эти объекты оттуда убираются, проходя через четыре возможных состояния.

Активное. Когда Активность на вершине стека, она видна и выходит на передний план, имея возможность принимать пользовательский ввод. Android будет пытаться любой ценой сохранить ее в рабочем состоянии, при необходимости прерывая работу других Активностей, которые находятся на более низких

позициях в стеке, обеспечивая для нее все необходимые ресурсы. Ее работа будет приостановлена, если на передний план выйдет другая Активность.

Приостановленное. В некоторых ситуациях ваша Активность будет видна на экране, но не сможет принимать пользовательский ввод: в этот момент она приостановлена. Такое состояние наступает, когда полупрозрачные или плавающие диалоговые окна становятся активными и частично ее перекрывают. В приостановленном виде Активность рассматривается как полноценно работающая, однако она не может взаимодействовать с пользователем. Ее работа может преждевременно завершиться, если системе нужно выделить ресурсы для той Активности, что на переднем плане. При полном исчезновении с экрана Активность останавливается.

Остановленное. Когда Активность перестает быть видимой, она останавливается и остается в памяти, сохраняя всю информацию о своем состоянии. То есть становится кандидатом на преждевременное закрытие, если системе понадобится память для чего-нибудь другого. При остановке Активности важно сохранить данные и текущее состояние пользовательского интерфейса. Как только объект Activity завершает свою работу или закрывается, он становится неактивным.

Неактивное. Это состояние наступает после того, как работа объекта Activity была завершена, и перед тем, как он будет запущен снова. Такая Активность удаляется из стека и должна запускаться повторно, чтобы ее можно было вывести на экран и снова использовать.

Смена состояний — недетерминированный процесс и полностью управляется системным диспетчером памяти. Сперва Android закроет приложения, содержащие объекты Activity в неактивном состоянии, потом перейдет к тем, чьи Активности остановлены или приостановлены (только в крайних случаях).

Порядок, в котором завершается работа процессов с целью освобождения ресурсов, определяется их приоритетами. Этот показатель берется из самого приоритетного компонента (приоритеты представлены на рисунке 1.6).

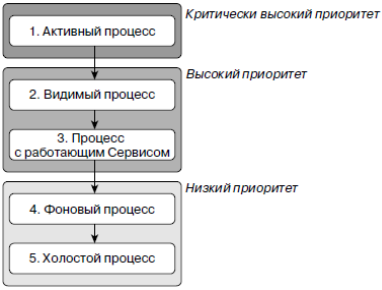

Рис. 1.6

Если приоритет двух приложений одинаковый, первым будет закрыто то, которое дольше всего проработало с пониженным приоритетом. На приоритет процесса также влияют межпрограммные связи. Допустим, одно приложение зависит от Сервиса или Источника данных, которые предоставляются другим приложением. Из этого следует, что у второго приложения приоритет как минимум не ниже, чем у первого. Любые приложения в Android продолжают работу и остаются в памяти до тех пор, пока системе не потребуются ресурсы для других программ. Важно, чтобы структура приложения была корректной, и чтобы его приоритет соответствовал работе, которую оно выполняет. В противном случае приложение может закрыться во время выполнения каких-то важных действий.

Активные процессы. Активные процессы (на переднем плане) содержат компоненты, взаимодействующие с пользователем. Android поддерживает их отзывчивость, освобождая дополнительные ресурсы. Как правило, таких процессов очень мало, и они закрываются в самую последнюю очередь.

Активные процессы включают в себя:

- Объекты Activity в активном состоянии, то есть те, которые находятся на переднем плане и отвечают на пользовательские события;
- Широковещательные приемники, обрабатывающие события с помощью методов onReceive;
- Сервисы, в которых запущены обработчики onStart, onCreate или onDestroy;
- Сервисы, предназначенные для работы на переднем плане.

Видимые процессы. Видимые, но не выполняющиеся в данный момент процессы содержат Активности, которые отображаются на экране. Речь о видимых Активностях, которые не находятся на переднем плане и не отвечают на пользовательские события. Это происходит, когда Активность частично перекрыта (диалоговым окном или другой полупрозрачной Активностью). Процессов, которые выводятся на экран, очень мало, поэтому их работа прерывается только в крайнем случае, если не хватает ресурсов для активных приложений.

Процессы с запущенными сервисами. Это процессы, содержащие работающие Сервисы. Компоненты Service могут выполняться непрерывно и не должны иметь графического интерфейса. Поскольку фоновые Сервисы не взаимодействуют с пользователем напрямую, они получают немного меньший приоритет, чем видимые Активности. Наличие таких Сервисов выводит процесс на передний план и делает его преждевременное завершение возможным, только если потребуются ресурсы для активных или видимых приложений.

Фоновые процессы. Процессы, не имеющие ни видимых Активностей, ни работающих Сервисов. Как правило, существует множество фоновых процессов, работа которых будет завершаться по принципу «последний запущенный закрывается первым», чтобы освободить ресурсы для приложений, работающих на переднем плане.

Холостые процессы. Для улучшения общей производительности системы Android часто сохраняет в памяти приложения, которые завершили жизненный цикл. Android поддерживает этот кэш, чтобы уменьшить время повторного запуска программ. Работа таких процессов прерывается при необходимости.

Для разработки приложений под Android вам нужно установить:

- JDK (Java Development Kit),
- IDE,
- Android SDK
- Android Development Tools.

Перед установкой программ поговорим о системных требованиях и материальной базе, необходимой для разработки Android-приложений. Прежде всего, вам нужен компьютер, причем совершенно неважно, под управлением какой операционной системы он будет работать. Ведь весь набор необходимых программ может работать под управлением Windows, Linux, Mac OS.

Какую операционную систему лучше использовать? Разработка Android-приложений не зависит от конкретной операционной системы, поскольку запуск и отладка Android-приложения будет осуществляться в эмуляторе мобильного устройства с поддержкой Android.

Что же касается версий операционных систем, то можно использовать Windows XP (не важно какой Service Pack), Windows Vista или Windows 7. Рекомендуется использовать 32-битную версию Windows 7.

Если вы предпочтете Linux, то желательно использовать последнюю или хотя бы предпоследнюю версию вашего дистрибутива. А пользователям Mac OS нужна операционная система версии 10.4.8 или более новая.

Теоретически, для разработки Android-приложений вам не нужен ни телефон, ни любое другое устройство, поскольку для отладки и запуска приложений будет использоваться эмулятор, входящий в состав

Android SDK, а при установке Android SDK можно будет выбрать версию платформы, для которой будет производиться разработка программ. Вы можете выбрать даже несколько версий, например 2.4 и 4.1-4.4 — самые востребованные версии, как показывает рынок.

Наличие физического устройства, хоть и не обязательно, но весьма желательно — для тестирования программы в реальных условиях. Эмулятор есть эмулятор, а реальное устройство может показать недочеты вашей программы, которые невозможно будет заметить в эмуляторе. Так что понадобится устройство с поддержкой Android той версии, под которую вы планируете разрабатывать программы. Если же планируется разработка программ под разные версии Android, желательно обзавестись несколькими устройствами, лучше разных производителей — ведь везде есть свои нюансы, а чем «разношерстнее» оборудование, тем больше вероятность возникновения всякого рода непредвиденных обстоятельств — то, что и нужно для процесса отладки программы.

Первым делом нужно установить Java Development Kit. Для запуска программ, написанных на Java, необходима среда выполнения Java — Java Runtime Environment (JRE). Данная среда во многих случаях уже установлена на вашем компьютере, поскольку необходима для выполнения некоторых программ, например, для популярного офисного пакета OpenOffice.org.

Для разработки Java-программ понадобится комплект разработчика Java-приложений — Java Development Kit, включающий компилятор Java, стандартные библиотеки классов Java, документацию, примеры и саму JRE, его можно скачать с официального сайта Oracle (рисунок 1.7).

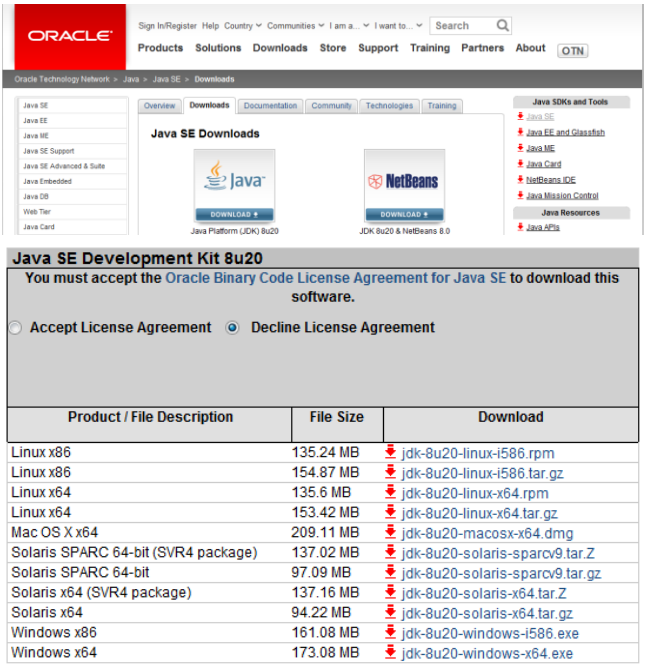

Рис. 1.7

После загрузки запустите скачанный файл. В процессе установки JDK нет ничего сложного, просто нажимайте кнопку Next и следуйте инструкциям мастера установки. Запомните каталог, в который вы установите JDK и JRE.

После установки JDK нужно установить IDE (Integrated Development Environment), например, Eclipse, скачать которую можно по адресу: http://www.eclipse.org/downloads/

На страничке загрузки Eclipse будет довольно много вариантов. Вам нужна версия Eclipse IDE for Java Developers. Версия Eclipse IDE for Java EE Developers не подходит, поскольку мы используем JDK SE, а не JDK EE.

При загрузке обратите внимание на архитектуру процессора и операционной системы: если у вас 32-разрядная версия Windows, не

нужно загружать 64-разрядную версию Eclipse, даже если процессор у вас 64-разрядный.

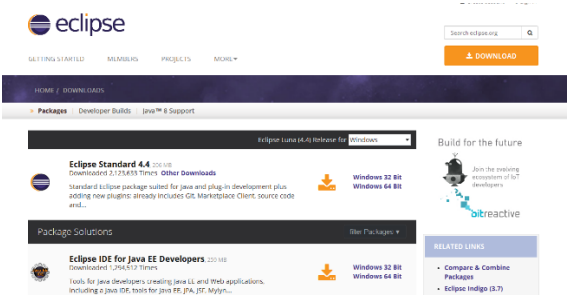

Рис. 1.8

Среда Eclipse распространяется не в виде инсталлятора, а в виде ZIP-архива. Распакуйте загруженный архив, перейдите в каталог eclipse (появится в результате распаковки архива) и запустите исполнимый файл eclipse.exe. После запуска Eclipse вам нужно будет выбрать каталог для рабочего пространства — каталог, в который Eclipse будет сохранять ваши проекты. После этого вы увидите основное окно Eclipse.

В дальнейшем необходимо будет связать Eclipse с Android Development Tools.

Почему именно Eclipse, а не популярная среда NetBeans? Во-первых, Eclipse является наиболее полно документированной средой разработки. А во-вторых, Google выпустила специальный плагин для Eclipse — Android Development Tools, облегчающий разработку Android-приложений в Eclipse. Данный плагин создает необходимую структуру для Android-проекта и автоматически устанавливает необходимые параметры компилятора.

Android Development Tools — это расширение для среды Eclipse, упрощающее разработку Android-приложений.

Чтобы установить ADT, запустите Eclipse и выберите команду меню Help | Install New Software. Появится окно, в котором нужно нажать кнопку Add. В следующем окне введите имя репозитория (можно ввести все, что угодно) и адрес — https://dl-ssl.google.com/android/eclipse

После этого вы вернетесь в окно Install и программа прочитает содержимое репозитория. Отметьте все компоненты и нажмите кпопку Next.

Далее, когда увидите сообщение о том, что все выбранные компоненты установлены, нажмите кнопку Next, установите переключатель I accept the terms of the license agreements и нажмите кпопку Finish

Если в процессе установки ADT вы увидите сообщение о неподписанном контенте, пусть это вас не сбивает с толку, просто нажмите кпопку OK.

Далее вы получите сообщение о необходимости перезагрузки Eclipse, согласитесь, нажав кнопку Restart Now.

Необходимо связать Eclipse с каталогом Android SDK. Для этого дождитесь повторного запуска Eclipse и в меню окна Eclipse выполните команду Window | Preferences. Перейдите в раздел Android и в поле SDK Location введите каталог, в который вы установили Android SDK (или выберите его с помощью кнопки Browse). Нажмите кнопку OK, после чего опять откройте окно настроек Eclipse и перейдите в раздел Android, чтобы убедиться, что поддерживаются все установленные ранее платформы Android. Платформа по умолчанию — 2.3.3 (рисунок 1.9).

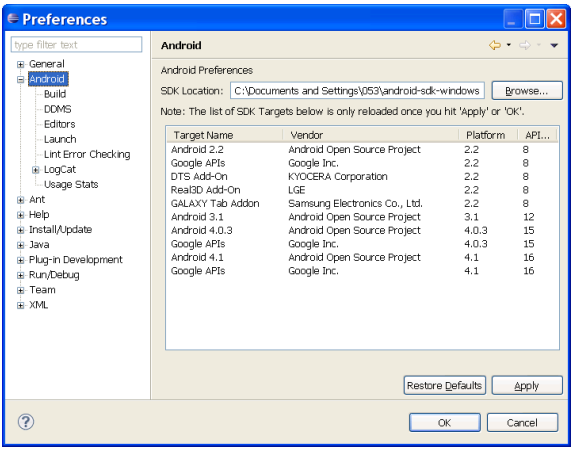

Рис. 1.9

Ранее было сказано, что плагин ADT упрощает разработку Android-приложений. Во-первых, ADT добавляет в Eclipse мастер создания проекта Android. Во-вторых, добавляет редактор Layout Editor, необходимый для создания графического интерфейса приложения. В-третьих, добавляет редакторы XML-ресурсов приложения. Также ADT позволяет запускать эмулятор Android и добавляет средства отладки непосредственно в среду Eclipse, что позволяет отлаживать программы, не выходя из привычной IDE.

Плагин добавляет в среду Eclipse следующие модули:

– мастер проектов Android, который помогает создавать новые проекты и включает стандартный шаблон приложения;
– основанный на формах манифест, шаблон и редактор ресурсов, которые помогают создавать, редактировать и проверять ваши XML-ресурсы;
– автоматическое построение проектов для Android, конвертацию исходных кодов в исполняемые файлы (.dex), упаковку в файлы пакетов (.apk) и установку пакетов в виртуальные машины Dalvik;

– менеджер виртуальных устройств Android, позволяющий создавать и управлять эмуляторами с виртуальными устройствами, на которых имитируется запуск специальных релизов ОС Android с присутствующими на устройстве ограничениями памяти;
– эмулятор Android с возможностью настройки его интерфейса и изменения установок сети, а также способностью имитировать входящиезвонки и SMS-сообщения;
– службу мониторинга отладки Dalvik (DDMS) с функцией перенаправления портов, стеком, кучей и возможностью просмотра потоков, информации о процессах и функцией снятия скриншотов;
– доступ к устройству или файловой системе эмулятора, благодаря чему можно перемещаться по дереву папок и передавать файлы;
– функцию отладки на этапе выполнения, благодаря чему можно устанавливать точки остановки и просматривать стеки вызовов;
– журнал событий Android и ВМ Dalvik, а также выводимый в консоль список сообщений.

Уровень API — это число, которое однозначно идентифицирует версию Android. Именно это число указывается напротив версии платформы в окне настроек Eclipse:

Version	Code name	Release date	API level	Distribution[a]
4.4	*KitKat*	October 31, 2013	19	20.9%
4.3	*Jelly Bean*	July 24, 2013	18	7.9%
4.2.x		November 13, 2012	17	19.8%
4.1.x		July 9, 2012	16	26.5%
4.0.3–4.0.4	*Ice Cream Sandwich*	December 16, 2011	15	10.6%
2.3.3–2.3.7	*Gingerbread*	February 9, 2011	10	13.6%
2.2	*Froyo*	May 20, 2010	8	0.7%

Рис. 1.10

Самая важная утилита из набора Android SDK — это эмулятор Android-устройства. Эмулятор позволяет отлаживать и тестировать приложения в реальной среде выполнения без необходимости их установки на физическое устройство. Даже если у вас есть физическое устройство, не спешите на нем запускать ваше приложение, которое может работать нестабильно. Сначала нужно запустить устройство в эмуляторе, если приложение будет работать нормально, можно попробовать запускать его на реальном устройстве.

Эмулятор сразу не готов к использованию. Перед его запуском нужно создать Android Virtual Device (AVD) — виртуальное устройство Android. AVD определяет настройки целевого устройства, вы можете создать несколько AVD, например, одно для платформы 2.2, другое — для 3.0. Одно с экраном 320x480, другое — 480x800, что позволяет тестировать приложения в разных условиях.

Создать AVD можно утилитой android.bat из каталога tools или визуально с помощью окна Android SDK and AVD Manager. В разделе Virtual device перечислены созданные вами AVD, по умолчанию ни одного виртуального устройства не создано. Для создания нового устройства нажмите кнопку New. В появившемся окне введите название устройства (Name), выберите целевую платформу (Target). При выборе платформы помните об обратной совместимости: если вы выбрали платформу 2.2, то эмулятор будет поддерживать SDK версий 2.0, 1.6 и 1.5 (более старые версии не поддерживаются текущей версией SDK). Тем не менее для платформ 2.x и 3.x рекомендовано создать разные AVD.

Параметр SD Card позволяет создать внешнюю карту устройства. Вы можете или указать размер новой карты (не менее 9 Мбайт).

Очень важный параметр Skin, позволяющий выбрать разрешение экрана устройства:

– **WVGA800** (Wide Video Graphics Array) — высокая плотность, нормальный (полноразмерный) экран, размер 480x800;

– **HVGA** (Half-size VGA Video Graphics Array) — средняя плотность, нормальный экран, размер 320x480;
– **WVGA854** (Wide Video Graphics Array) — высокая плотность, нормальный экран, разрешение 480x854;
– **QVGA** (Quarter Video Graphics Array) — низкая плотность, разрешение 240x320, малый экран (как у Mini-версий смартфонов);
– **WQVGA** (Wide Quarter Video Graphics Array) — низкая плотность, нормальный экран, разрешение 240x400.

Выбирайте разрешение, которое поддерживает ваше реальное устройство (или то устройство, для которого вы хотите разработать программу).

Параметр Hardware содержит список оборудования смартфона (планшета), который будет эмулироваться. При желании можно нажать кнопку New и добавить новое устройство.

Для создания AVD нажмите кнопку Create AVD, после чего вы получите отчет о создании AVD.

Созданное вами устройство появится в списке Virtual devices.

Для запуска эмулятор выделите нужное вам виртуальное устройство и нажмите кнопку Start. Появится окно, в котором можно установить дополнительные параметры виртуального устройства, но, как правило, достаточно просто нажать кнопку Launch.

Эмулятор, тем не менее, не поддерживает некоторые функциональности, доступные на реальных устройствах:

– входящие и исходящие сообщения. Однако можно моделировать обращения по телефону через интерфейс эмулятора;
– соединение через USB;
– видеокамера (однако есть имитатор работы видеокамеры);
– подключение наушников;
– определение статуса соединения;

– определение уровня заряда аккумуляторной батареи;
– определение вставки или изъятия карты памяти;
– соединение по Bluetooth.

Конечно, реальные телефоны несколько отличаются от эмулятора, но в целом AVD разработан очень качественно и близок по функциональности к реальному устройству.

Нужно отметить, что запуск эмулятора — процесс очень длительный. Сначала в окне эмулятора будет красоваться текстовая надпись ANDROID, затем логотип Android и только спустя несколько минут вы увидите сам эмулятор. Внешний вид эмулятора отличается в зависимости от выбранной платформы и параметра Skin виртуального устройства.

Оставьте окно эмулятора открытым, пока открыто окно Eclipse. В следующий раз, когда вы запустите программу на Android, Eclipse поймет, что эмулятор уже готов, и просто отправит на него новую программу для запуска.

Каждое приложение для Android функционирует в отдельном процессе внутри собственного экземпляра машины Dalvik. Вся ответственность за память и управление процессами возлагается на Android, который останавливает или убивает процессы, если нужно освободить ресурсы. Dalvik и Android находятся на вершине ядра Linux, которое занимается низкоуровневым взаимодействием с аппаратным обеспечением, включая работу драйверов и управление памятью. При этом набор встроенного API позволяет получить доступ ко всем службам, функционалу и аппаратной начинке.

Компоновка — это **архитектура расположения элементов интерфейса** пользователя для конкретного окна, представляющего Activity. Компоновка определяет структуру расположения элементов в окне и содержит все элементы, которые предоставляются пользователю программы.

Эта важная тема, поскольку проектирование пользовательского интерфейса для мобильных телефонов **сложнее**, чем для настольных систем или для Web-страниц. Экраны мобильных телефонов имеют гораздо меньшее разрешение, чем обычные мониторы. Кроме того, существует **много разновидностей дисплеев** для мобильных телефонов, отличающихся размерами, разрешением и плотностью пикселов.

Необходимо также учесть, что большинство экранов для мобильных телефонов **сенсорные**, причем могут быть разного типа. Например, емкостный экран реагирует на касание пальцем, а для взаимодействия с резистивным экраном используется стилус. Поэтому важно правильно задавать компоновку и размеры элементов управления, чтобы пользователю было удобно управлять вашим приложением независимо от типа экрана.

Компоновка— это **архитектура расположения элементов интерфейса** пользователя для конкретного окна, представляющего Activity. Компоновка определяет структуру расположения элементов в окне и содержит все элементы, которые предоставляются пользователю программы.

Эта важная тема, поскольку проектирование пользовательского интерфейса для мобильных телефонов **сложнее**, чем для настольных систем или для Web-страниц. Экраны мобильных телефонов имеют гораздо меньшее разрешение, чем обычные мониторы. Кроме того, существует **много разновидностей дисплеев** для мобильных телефонов, отличающихся размерами, разрешением и плотностью пикселов.

Необходимо также учесть, что большинство экранов для мобильных телефонов **сенсорные**, причем могут быть разного типа. Например, емкостный экран реагирует на касание пальцем, а для взаимодействия с резистивным экраном используется стилус. Поэтому важно правильно задавать компоновку и размеры элементов управления, чтобы пользователю было удобно управлять вашим приложением независимо от типа экрана.

В Android-приложении графический интерфейс пользователя формируется с использованием объектов **view** и **viewGroup**. Класс view является базовым классом для viewGroup и состоит из коллекции объектов view (рисунок 2.1). Есть множество типов view и viewGroup, каждый из которых является потомком класса view.

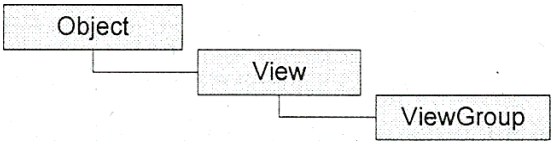

Рис. 2.1

Объекты view — это основные модули для создания графического интерфейса пользователя на платформе Android. Класс view служит базовым для классов элементов управления, называемых виджетами, — текстовых полей, кнопок и т. д.

Объект view — структура, свойства которой сохраняют параметры компоновки и содержание для определенной прямоугольной области экрана. Как объект в интерфейсе пользователя объект view является точкой взаимодействия пользователя и программы.

Класс **viewGroup** представляет собой контейнер, который служит ядром для подклассов, называемых **компоновками** (layouts). Эти классы формируют расположение элементов пользовательского

интерфейса на форме и содержат дочерние элементы View или ViewGroup (рисунок 2.2).

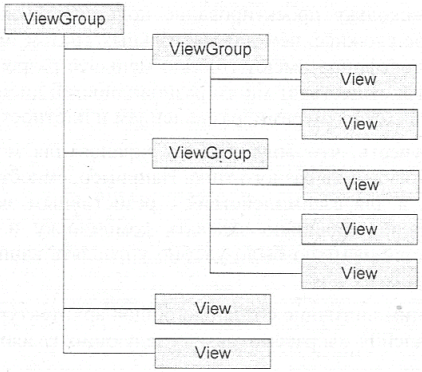

Рис. 2.2

На платформе Android необходимо определить **пользовательский интерфейс для каждого Activity**, используя иерархии узлов view и viewGroup, как показано на рис. Это **дерево иерархии** может быть и простым, и сложным — в зависимости от требований к графическому интерфейсу приложения.

Каждый элемент файла разметки является объектом класса View или ViewGroup. Если представить все элементы интерфейса пользователя в виде иерархии, то объекты класса ViewGroup будут ветвями дерева, а объекты класса View — листьями. Иерархия созданного интерфейса отображается на вкладке **Outline редактора интерфейса.**

При запуске программы система Android получает ссылку на корневой узел дерева и использует ее для прорисовки графического интерфейса на экране мобильного устройства. Система также анализирует элементы дерева от вершины дерева иерархии, прорисовывая дочерние объекты view и viewGroup и добавляя их родительским элементам. Для этого в методе **onCreate()** необходимо

вызвать метод **SetContentView()**, передав ему в качестве параметра ссылку на ресурс компоновки в следующем виде:

R.layout.layout_file_name

Например, если компоновка находится в файле **activity_main.xml**, ее загрузка в методе onCreate() происходит так, как представлено в листинге:

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

Прорисовка начинается с корневого узла дерева компоновки. Затем последовательно прорисовываются дочерние объекты дерева компоновки. Это означает, что родители будут прорисовываться раньше, чем их дочерние объекты, — т.е. по окончании процесса прорисовки родители будут находиться на заднем плане по отношению к дочерним узлам.

Компоновку можно объявлять **двумя способами**:

объявить элементы пользовательского интерфейса в XML-файле. Android обеспечивает прямой **XML-словарь**, который соответствует классам view и viewGroup;

создать компоновку для окна **в коде программы** во время выполнения — инициализировать объекты Layout и дочерние объекты View, ViewGroup и управлять их свойствами программно.

При создании пользовательского интерфейса можно использовать каждый из этих методов в отдельности или оба сразу для объявления и управления пользовательским интерфейсом в приложении. Например, можно объявить заданную по умолчанию компоновку окна вашего приложения в XML-файле, включая экранные элементы, которые появятся в них, и их свойства, а затем добавить код в приложение,

который во время выполнения изменит состояние объектов на экране, включая объявленные в XML-файле.

Самый общий способ определять компоновку и создавать иерархию элементов интерфейса — в **XML-файле компоновки**. XML предлагает удобную структуру для компоновки, похожую на HTML-компоновку Web-страницы.

Преимущество объявления пользовательского интерфейса в XML-файле состоит в том, что это дает возможность **отделить дизайн приложения от программного кода**, который управляет поведением приложения. Ваше описание пользовательского интерфейса является внешним по отношению к программному коду, это означает, что вы можете изменять пользовательский интерфейс в файле компоновки без необходимости изменения вашего программного кода.

Используя **XML-словарь Android**, можно быстро проектировать пользовательский интерфейс компоновки и экранные элементы, которые он содержит, тем же самым способом, которым вы создаете Web-страницы в HTML — с рядом вложенных элементов.

Каждый файл компоновки должен содержать **только один корневой элемент**, который должен быть объектом view или viewGroup. Как только вы определили корневой элемент, вы можете **добавить дополнительные объекты компоновки** или виджеты как дочерние элементы, чтобы постепенно формировать иерархию элементов, которую определяет создаваемая компоновка.

В каждом файле разметки должен быть только один корневой элемент. В нашем случае таким элементом является **LinearLayout** (линейная разметка). После определения корневого элемента вы можете добавить в него дополнительные объекты разметки или виджеты в качестве дочерних элементов.

Рассмотрим **атрибуты** элемента LinearLayout:

1. **xmlns:android** — объявление пространства имен Android. Стандартный атрибут и стандартное значение для

Android-приложения. Декларация пространства имен XML, которая сообщает среде Android, что вы ссылаетесь на общие атрибуты, определенные в пространстве имен Android. В каждом файле компоновки у корневого элемента должен быть атрибут со значением http://schemas.android.com/apk/res/android

2. **android:layout_width** — атрибут определяет, сколько из доступной ширины на экране должен использовать этот объект view (или viewGroup). В нашем случае он — единственный объект, таким образом, можно растянуть его на весь экран, которому в данном случае соответствует значение **fill_parent**;

3. **android:layout_height** — аналогичен атрибуту android:layout_width за исключением того, что он ссылается на доступную высоту экрана;

4. **android:text** — текст, который должен отобразить объект TextView. В нашем примере вместо строковой константы используется значение из файла strings.xml — строка hello. Благодаря такому приему очень легко выполнить локализацию приложения (перевод на другой язык).

У каждого объекта View или ViewGroup свой набор атрибутов.

ADT-плагин для Eclipse предлагает удобный инструмент — **визуальный редактор компоновки Layout Editor**, который применяется для создания и предварительного просмотра создаваемых файлов компоновки, которые находятся в каталоге res/layout/ проекта.

Например, можно создавать XML-компоновки для различных ориентаций экрана мобильного устройства (portrait, landscape), размеров экрана и языков интерфейса. Дополнительно, объявление компоновки в XML-файле облегчает визуализацию структуры вашего пользовательского интерфейса, что упрощает отладку приложения.

На вкладке **Outline** отображается компоновка в виде дерева. Каждый элемент в XML является объектом view или viewGroup (или его потомком). Объекты view — листья дерева, объекты viewGroup —

ветви. Вы можете также создавать объекты view и viewGroup в Java-коде, используя метод addview(view), чтобы динамически вставлять новые объекты view и viewGroup в существующую компоновку.

Используя различные виды viewGroup, можно структурировать дочерние объекты view и viewGroup многими способами в зависимости от требований к графическому интерфейсу приложения.

Для создания окон существует несколько стандартных типов компоновок, которые вы можете использовать в создаваемых приложениях:

– FrameLayout;
– LinearLayout;
– TableLayout;
– RelativeLayout

Все эти компоновки являются подклассами viewGroup и наследуют свойства, определенные в классе view (рисунок 2.3).

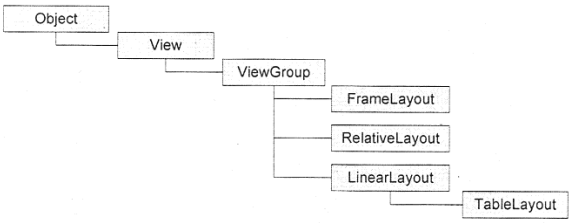

Рис. 2.3

Каждый, из этих типов компоновки предлагает уникальный набор параметров, которые используются, чтобы определить позиции дочерних элементов view и структуру компоновки на экране. В зависимости от требований, предъявляемых к пользовательскому интерфейсу, выбирается наиболее подходящий тип компоновки. Далее мы рассмотрим все варианты компоновок и их использование.

FrameLayout является **самым простым типом компоновки**. Это в основном пустое пространство на экране, которое можно позже **заполнить только единственным дочерним объектом** view или ViewGroup. Все дочерние элементы FrameLayout прикрепляются к **верхнему левому углу экрана**.

В компоновке FrameLayout нельзя определить различное местоположение для дочернего объекта view. Последующие дочерние объекты view будут просто **рисоваться поверх предыдущих**, частично или полностью затеняя их, если находящийся сверху объект непрозрачен, поэтому единственный дочерний элемент для FrameLayout обычно растянут до размеров родительского контейнера и имеет атрибуты `layout_width="fill_parent"` и `layout_height="fill_parent"`.

Компоновка FrameLayout **применяется довольно редко**, т.к. не позволяет создавать сложные окна с множеством элементов. Эту компоновку обычно используют для создания оверлеев. Например, если у вас в окне выводится изображение, занимающее весь экран (это может быть карта или картинка с видеокамеры), можно сверху на изображении расположить элементы управления (в дочернем контейнере, если их несколько), например, кнопки для управления камерой и рамку видоискателя, а также выводить индикацию времени съемки и другую полезную информацию.

FrameLayout — самый простой тип разметки. Все дочерние элементы FrameLayout будут прикреплены к верхнему левому углу экрана.

Единственное применение для FrameLayout — это ее использование внутри ячейки таблицы, а для создания полноценной разметки приложения данный вариант не годится.

Компоновка LinearLayout выравнивает все дочерние объекты view в одном направлении — вертикально или горизонтально, в зависимости от того, как определен атрибут ориентации **android:orientation:**

```
android:orientation="horizontal"
```
или
```
android:orientation="vertical"
```

Все дочерние элементы помещаются в стек один за другим, так что вертикальный список объектов view будет иметь только один дочерний элемент в строке независимо оттого, насколько широким он является. Горизонтальное расположение списка будет размещать элементы в одну строку с высотой, равной высоте самого высокого дочернего элемента списка.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button1" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button2" />
    <Button
        android:id="@+id/button3"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button3" />
```

```
</LinearLayout>
```

Обратите внимание, что у первых двух кнопок атрибуту android:layout_width присвоено значение wrap_content, а у третьей кнопки — fill_parent, т.е. последняя кнопка заполнит оставшееся свободное пространство в компоновке.

В результате получится линейное горизонтальное размещение дочерних элементов. Если изменить в корневом элементе значение атрибута `android:orientation=”vertical”`, элементы в контейнере расположатся вертикально.

Компоновка LinearLayout также поддерживает атрибут android: layout_weight, который назначает **индивидуальный вес для дочернего элемента**. Этот атрибут определяет "важность" объекта view и позволяет этому элементу расширяться, чтобы заполнить любое оставшееся пространство в родительском объекте view. **Заданный по умолчанию вес является нулевым**.

Например, если есть три текстовых поля, и двум из них объявлен вес со значением 1, в то время как другому не дается никакого веса (0), третье текстовое поле без веса не будет расширяться и займет область, определяемую размером текста, отображаемого этим полем. Другие два расширятся одинаково, чтобы заполнить остаток пространства, не занятого третьим полем. Если третьему полю присвоить вес 2 (вместо 0), это поле будет объявлено как "более важное", чем два других, так что третье поле получит 50% общего пространства, в то время как первые два получат по 25% общего пространства.

```
<Button
   android:id="@+id/button1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Button1"
   android:layout_weight="0"/>
```

```
<Button
   android:id="@+id/button2"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Button2"
   android:layout_weight="0"/>
<Button
   android:id="@+id/button3"
   android:layout_height="wrap_content"
   android:layout_width="wrap_content"
   android:text="Button3"
   android:layout_weight="2"/>
```

Рис. 2.4

```
<Button
   android:id="@+id/button1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Button1"
   android:layout_weight="0"/>
<Button
  android:id="@+id/button2"
  android:layout_width="wrap_content"
```

```
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="Button2" />
   <Button
      android:id="@+id/button3"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:text="Button3"
      android:layout_weight="2"/>
</LinearLayout>
```

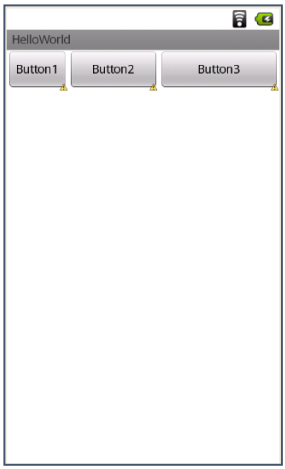

Рис. 2.5

```
   <Button
      android:id="@+id/button1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Button1"
      android:layout_weight="0"/>
   <Button
     android:id="@+id/button2"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
```

```
      android:layout_weight="5"
      android:text="Button2" />
   <Button
      android:id="@+id/button3"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:text="Button3"
      android:layout_weight="2"/>
</LinearLayout>
```

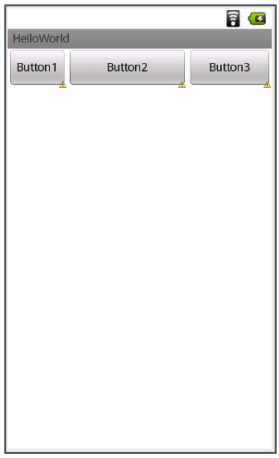

Рис. 2.6

Компоновки могут быть и вложенными. При проектировании окон с многочисленными элементами управления для заданного расположения элементов часто задаются вложенные компоновки, которые являются контейнерами для элементов управления. Например, для корневой LinearLayout с атрибутом orientation="vertical" мы можем задать простой дочерний элемент Button и еще два контейнера LinearLayout с атрибутом orientation="horizontal", которые, в свою очередь, содержат дочерние элементы управления.

Такое применение вложенных компоновок позволяет строить гибкие и легко перенастраиваемые окна и является **самым**

распространенным способом при создании пользовательского интерфейса для Android-приложений.

Компоновка TableLayout позиционирует свои дочерние элементы в строки и столбцы. TableLayout не отображает линии обрамления для их строк, столбцов или ячеек. TableLayout может иметь строки с разным количеством ячеек. При формировании компоновки таблицы некоторые ячейки при необходимости можно оставлять пустыми.

При создании компоновки для строк используются объекты **TableRow**, которые являются дочерними классами TabieLayout (каждый TableRow определяет единственную строку в таблице). Строка может не иметь ячеек или иметь одну и более ячеек, которые являются контейнерами для других объектов view или viewGroup. Ячейка может также быть объектом viewGroup (например, допускается вложить другой TabieLayout или LinearLayout как ячейку).

Для примера с использованием компоновки TableLayout можно создать окно, похожее на наборную панель телефона с 12 кнопками. Рассмотрим пример создания TableLayout с четырьмя дочерними TableRow и двенадцатью кнопками, по три кнопки в каждой строке.

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:id="@+id/TableLayout01"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_height="fill_parent" >
<TableRow
  android:id="@+id/TableRow0l"
  android:layout_height="wrap_content"
  android:layout_width="fill_parent"
  android:gravity="center">
  <Button
```

```
      android:id="@+id/Button01"
      android:layout_height="wrap_content"
      android:text="1"
      android:layout_width="20pt"/>
   <Button
      android:id="@+id/Button02"
      android:layout_height="wrap_content"
      android:text="2"
      android:layout_width="20pt"/>
   <Button
      android:id="@+id/Button03"
      android:layout_height="wrap_content"
      android:text="3"
      android:layout_width="20pt"/>
</TableRow>
```

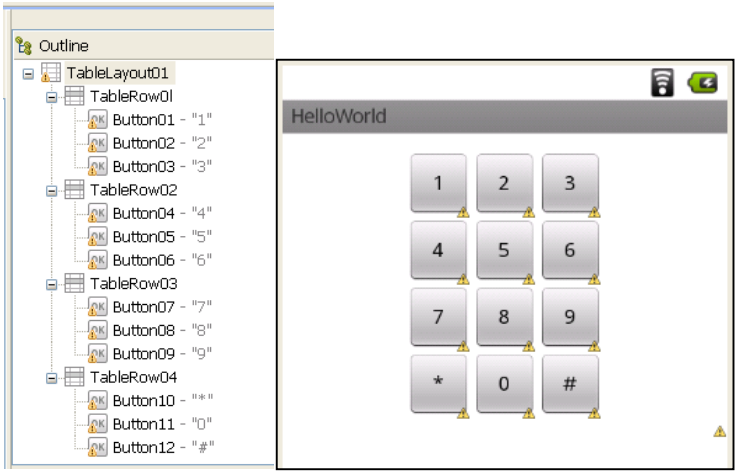

Рис. 2.7

Компоновка TableLayout на практике применяется довольно редко, обычно вместо нее используют сочетание компоновок LinearLayout. TableLayout удобно использовать, если расположение элементов представлено в виде "таблицы“.

Когда вы вникните в структуру XML-файла, вы обнаружите, что редактировать разметку вручную даже проще, чем использовать интерфейс Eclipse. Во всяком случае, создать разметку для Android-приложения вручную не сложнее, чем создать HTML-страницу. Предпочтительнее создавать разметку вручную, а устанавливать некоторые свойства — с помощью редактора разметки — не всегда удается запомнить все необходимые свойства и их значения.

RelativeLayout (относительная компоновка) позволяет дочерним объектам **определять свою позицию относительно родительского объекта, или относительно соседних дочерних элементов** (по идентификатору элемента).

В RelativeLayout дочерние элементы расположены так, что если первый элемент расположен по центру экрана, другие элементы, выровненные относительно первого элемента, будут выровнены относительно центра экрана. При таком расположении, при объявлении компоновки в XML-файле, элемент, на который будут **ссылаться для позиционирования другие объекты**, должен быть объявлен раньше, чем другие элементы, которые **обращаются к нему по его идентификатору**.

Если в программном коде мы не работаем с некоторыми элементами пользовательского интерфейса, создавать идентификаторы для них необязательно, однако определение идентификаторов для объектов важно при создании RelativeLayout. В компоновке RelativeLayout расположение элемента может определяться относительно другого элемента, на который ссылаются через его уникальный идентификатор:

```
android:layout_toLeftOf="@id/TextView1"
```

Тип компоновки RelativeLayout применяется довольно редко. Тем более что такое задание расположения элементов зависит от

разрешения и ориентации экрана мобильного устройства. Если вы будете использовать RelativeLayout в собственных приложениях, всегда проверяйте внешний вид окна для различных разрешений и ориентации экрана, поскольку возможно наложение элементов друг на друга.

В состав Android SDK входит утилита **Hierarchy Viewer** — полезный инструмент при **разработке окон со сложной компоновкой**. Он позволяет отладить и оптимизировать пользовательский интерфейс приложений. Hierarchy Viewer отображает визуальное представление иерархии элементов создаваемой компоновки и экранную лупу для просмотра пиксельной структуры компоновки экрана мобильного устройства — Pixel Perfect View.

Для запуска Hierarchy Viewer необходимо выполнить следующие действия: подключить мобильное устройство или запустить эмулятор Android, затем ввести в командной строке **hierarchyviewer** (или запустить файл hierarchyviewer.bat из каталога tools/ в Android SDK).

Выберите нужное окно для отладки и нажмите кнопку **Load View Hierarchy**. Откроется окно **Layout View.**

Окно **Layout View** в левой части отображает иерархию компоновки и ее свойства. У окна справа есть три панели:

Tree View — дерево элементов;

Properties View — список свойств выбранного элемента;

Wire-frame View — каркас компоновки.

Для отображения свойств элемента на панели **Properties View** необходимо выбрать элемент в дереве компоновки на панели **Tree View.** При выборе элемента на панели **Wire-frame View** будут в красном прямоугольнике показаны границы этого элемента. Двойной щелчок на узле дерева компоновки открывает новое окно с рендерингом этого элемента.

Чтобы переключаться между окнами, используются также три кнопки на панели состояния. Если нажать кнопку **Inspect Screenshot,** откроется окно **Pixel Perfect View,** которое предоставляет экранную лупу для детального просмотра компоновки.

Лекция 1.2. МЕНЮ И ВИДЖЕТЫ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА

Меню в Android представляют собой **объекты Java**, однако они представляются и **как ресурсы**. А поскольку меню — это ресурсы, Android SDK позволяет **загружать их из XML-файлов**, как и все другие ресурсы, генерируя идентификаторы ресурсов для каждого из загруженных пунктов меню.

Основой поддержки меню в Android является класс **android.view.Menu**. Каждое действие в Android связано с одним объектом меню, который содержит **ряд пунктов и подменю**.

Пункты меню представлены классом **android.view.MenuItem**, а подменю — классом **android.view.SubMenu**. Эти взаимосвязи графически изображены на рисунке. Строго говоря, на этом рисунке представлена не диаграмма классов, а структурная диаграмма для визуализации взаимосвязей между различными классами и функциями, которые имеют дело с меню.

У пункта меню есть имя (название), **идентификатор** пункта меню, **порядок** сортировки (в SDK называется просто "order" — упорядоченность) и идентификатор (или номер). Эти идентификаторы позволяют задать порядок пунктов в меню. Например, если у одного пункта меню порядок сортировки равен 4, а у другого — 6, то первый пункт будет находиться в меню выше второго.

Пункты меню можно объединять в группы, назначая им одинаковый идентификатор группы, который является атрибутом объекта пункта меню. Несколько пунктов меню с одинаковым идентификатором группы считаются входящими в одну группу. На рис. показаны **два метода обратного вызова**, которые позволяют создавать пункты меню и реагировать на их выбор: **onCreateOptionsMenu** и **onOptionsItemSelected**.

Android SDK позволяет не создавать объекты меню с нуля. Поскольку с каждым меню связывается действие, Android создает это

меню для данного действия и передает его методу обратного вызова onCreateOptionsMenu из класса действия. (Как понятно из имени метода, меню в Android называются также *меню выбора* (options menu)) С помощью этого метода можно заполнить набором пунктов одно переданное меню.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
   // Вызов базового класса для включения системных меню
   super.onCreateOptionsMenu(menu);
   menu.add(0 // Группа
    , 1 // Идентификатор элемента
    , 0 // Порядок
    , "append"); // Название
   menu.add(0, 2, 1, "item2");
   menu.add(0, 3, 2, "clear");
   // Для видимости меню важно возвратить true
   return true;
}
```

Первый параметр при добавлении пункта меню является **идентификатором группы** (целое число). Второй параметр — **идентификатор пункта меню**, который возвращается функции обратного вызова при выборе этого пункта. Третий аргумент представляет **идентификатор порядка**.

Последний аргумент — **имя, или название, пункта меню**. Вместо обычного текста здесь можно использовать строковый ресурс из файла констант R.java. Идентификаторы группы, пункта меню и порядка указывать не обязательно; вместо них можно задать значение Menu.NONE.

Существует **несколько способов** реагирования на щелчки на пунктах меню в Android:

это метод **onOptionsItemSelected** из класса действия, обособленные слушатели и намерения. В данном разделе будут рассмотрены все эти техники.

Реагирование на выбор пунктов меню с помощью метода onOptionsItemSelected

При выборе пункта меню Android вызывает метод обратного вызова **onOptionsItemSelected** из класса Activity

Основное здесь — **проверка идентификатора** пункта меню с помощью метода **getItemld()** класса MenuItem с последующим выполнением необходимых действий. Если метод onOptionsItemSelected () обрабатывает пункт меню, он возвращает true, и событие меню не будет распространяться дальше. А для обратных вызовов для пунктов меню, которые метод onOptionsItemSelected () не обрабатывает, должен вызываться родительский метод super.onOptionsItemSelected. Стандартная реализация onOptionsItemSelected () возвращает false, чтобы была выполнена нормальная обработка, к которой относятся и альтернативные средства выполнения обработки щелчков на пунктах меню.

Операционная система Android предлагает **три вида меню**:

OptionsMenu — меню выбора опций, появляется внизу экрана при нажатии кнопки Menu на мобильном устройстве;

ContextMenu — контекстное меню, появляется при долгом касании (2 или более секунды) сенсорного экрана;

SubMenu — подменю, привязывается к конкретному пункту меню (меню выбора опций или контекстному меню). Пункты подменю не поддерживают вложенного меню.

В случае с OptionsMenu существует два типа меню:

IconMenu — меню со значками, добавляет значки к тексту в пункты меню. Это единственный тип меню, поддерживающий значки. В IconMenu может быть максимум шесть пунктов;

ExpandedMenu — расширенное меню, представляет собой вертикальный выпадающий список меню. Появляется при наличии в меню более шести пунктов, при этом в меню появляется дополнительный пункт **More**. Расширенное меню добавляется автоматически самой операционной системой. При нажатии **More** появятся дополнительные пункты меню, которые не поместились в основной части меню.

Начнем с создания самого часто использующегося меню — меню выбора опций, которое появляется, когда пользователь нажмет кнопку **Menu** на мобильном устройстве.

Меню можно создать в файле разметки или с помощью метода add(). Мы будем использовать второй способ, оставив файл разметки без изменений.

Первым делом нужно определить идентификаторы создаваемых пунктов меню, делается это так:

```
public static final int IDM_NEW = 101;
public static final int IDM_OPEN = 102;
public static final int IDM_SAVE = 103;
public static final int IDM_EXIT = 104;
```

В нашем меню будут четыре пункта с идентификаторами IDM_NEW, IDM_OPEN, IDM_SAVE, IDM_EXIT.

Далее для каждого пункта меню нужно вызвать метод add():

```
menu.add(Menu.NONE, IDM_NEW, Menu.NONE, "New game")
   .setAlphabeticShortcut('n');
```

В данном случае мы не только добавляем новый пункт меню, но и устанавливаем для него клавишу быстрого доступа. Если клавиша быстрого доступа не нужна, то оператор добавления пункта меню можно записать короче:

```
menu.add(Menu.NONE, IDM_NEW, Menu.NONE, "New game");
```

Методу add() нужно передать **четыре параметра**:

- **идентификатор группы меню** — позволяет связать данный пункт меню с группой других пунктов этого меню. Используется для создания сложных меню. В нашем случае можно воспользоваться значением Menu.NONE, потому что идентификатор группы задавать не нужно;
- **идентификатор меню** — позволяет однозначно идентифицировать пункт меню, далее идентификатор поможет определить, какой пункт меню был выбран пользователем;
- **порядок расположения пункта в меню** — по умолчанию пункты размещаются в меню в порядке их добавления методом add(), поэтому в качестве значения этого параметра можно тоже указать Menu.NONE;
- **заголовок** — задает заголовок пункта меню, видимый пользователем. Вы можете задать как текстовую константу, так и указать строковый ресурс.

Пользователи, работающие на настольных компьютерах, без сомнения, знакомы с контекстными меню. Например, в Windows-приложениях контекстные меню открываются в результате щелчка правой кнопкой мыши на элементе пользовательского интерфейса. Android поддерживает такую же идею с помощью **операции *длинного щелчка* (long click).** Длинный щелчок — это щелчок кнопкой мыши на любом представлении Android, который длится несколько дольше, чем обычно.

В карманных устройствах, таких как мобильные телефоны, щелчки кнопкой мыши имитируются целым рядом способов, в зависимости от механизма навигации. Если у телефона имеется колесико для перемещения курсора, то нажатие на это колесико выполняет функцию

щелчка кнопкой мыши. Если на устройстве имеется сенсорная панель, то эквивалентом щелчка является постукивание или нажатие. Независимо от реализации щелчка кнопкой мыши на конкретном устройстве, более долгое удерживание щелчка считается длинным щелчком.

Контекстные меню структурно отличается от стандартных меню, и для них характерны некоторые нюансы, не присущие меню выбора. На рис. показано, что контекстное меню представлено **классом ContextMenu** в архитектуре меню Android. Как и Menu, класс ContextMenu может содержать ряд пунктов меню. Для добавления пунктов в контекстное меню используется тот же набор методов, что и в классе Menu.

Наибольшее различие между Menu и ContextMenu состоит во владельце меню. **Владельцами обычных меню выбора являются действия, а владельцами контекстных меню являются представления.** Это понятно, т.к. длинные щелчки, активизирующие контекстные меню, применяются к *представлению,* на котором выполнен щелчок. Поэтому действие может иметь только одно меню выбора, но несколько контекстных меню: ведь действие может содержать несколько представлений, а каждое представление может иметь собственное контекстное меню, и поэтому действие может иметь столько контекстных меню, сколько в нем представлений.

Несмотря на то что владельцем контекстного меню является представление, **метод для заполнения контекстных меню находится в классе Activity**. Этот метод называется activity.onCreateContextMenu (), и его функции аналогичны методу activity.onCreateOptionsMenu () . Этот метод обратного вызова также несет информацию о представлении, для которого нужно заполнить пункты контекстного меню.

С контекстными меню связан еще один существенный момент. Метод onCreateOptionsMenu () автоматически вызывается для каждого

действия, но в отношении метода onCreateContextMenu () это не так. Представление в действии не обязано иметь контекстное меню.

Например, в действии может быть три представления, но программист может активизировать контекстное меню лишь для одного из них. Если понадобится, чтобы у какого-либо представления было контекстное меню, это представление необходимо зарегистрировать вместе с его действием специально для обретения контекстного меню. Для этого предназначен метод activity.registerForContextMenu (view).

Создать контекстное меню чуть сложнее, чем обычное. Ведь нужно не только создать меню, но и привязать его к определенному объекту интерфейса программы — ведь это контекстное меню, и появляться оно должно не просто так, а при долгом нажатии на определенный объект.

Шаги по реализации контекстного меню

1. Зарегистрируйте представление для контекстного меню в методе onCreate() нужного действия.
2. Заполните контекстное меню с помощью метода onCreateContextMenu().
3. Обработайте щелчки на пунктах контекстного меню.

Подменю можно добавить в любое другое меню, кроме контекстного. Подменю полезно при создании очень сложных приложений, где пользователю доступно много функций.

До сих пор мы создавали все наши меню программным образом. Конечно, это утомительное занятие, т.к. для каждого меню приходится указывать несколько идентификаторов и определять константы для каждого такого идентификатора.

Но вместо этого можно определять меню с помощью XML-файлов — в Android меню также являются ресурсами. Такой подход к созданию меню имеет несколько преимуществ: возможность именования меню, автоматическое их упорядочение и присваивание

идентификаторов. Кроме того, для текста меню поддерживается локализация.

Для работы с XML-меню нужно выполнить следующие шаги.

1. Определите XML-файл с дескрипторами меню.

2 Поместить этот файл в подкаталог /res/menu. Имя файла может быть произвольным, и файлов может быть сколько угодно. Android автоматически генерирует идентификатор ресурса для такого файла меню.

3. Воспользуйтесь идентификатором ресурса для файла меню, чтобы загрузить XML-файл в меню.

4. Описать реагирование на пункты меню, используя идентификаторы ресурсов, сгенерированные для каждого пункта.

Приложения могут отображать два типа уведомлений: краткие всплывающие сообщения (**Toast Notification**) и постоянные напоминания (**Status Bar Notification**). **Первые** отображаются на экране мобильного устройства какое-то время и не требуют внимания пользователя. Как правило, это не критические информационные уведомления. **Вторые** постоянно отображаются в строке состояния и требуют реакции пользователя.

Например, приложение требует подключения к вашему серверу. Если соединение успешно установлено, можно отобразить краткое уведомление, а вот если подключиться не получилось, тогда отображается постоянное уведомление, чтобы пользователь сразу мог понять, почему приложение не работает.

Чтобы отобразить всплывающее сообщение, используйте класс **Toast** и его методы **makeText** (создает текст уведомления) и **Show** (отображает уведомление):

```
Context context = getApplicationContext();
Toast toast = Toast.makeText(context,
   "This is notification",
```

```
Toast.LENGTH_SHORT);
```

Первый параметр метода makeText — это контекст приложения, который можно получить с помощью вызова getApplicationContext().

Второй параметр — текст уведомления.

Третий — задает продолжительность отображения уведомления:

LENGTH_SHORT — небольшая продолжительность (1-2 секунды) отображения текстового уведомления;

LENGTH_LONG — показывает уведомление в течение более длительного периода времени (примерно 4 секунды).

Создать уведомление в строке состояния немного сложнее. Для этого нужно использовать два класса: **Notification** и **NotificationManager**. Первый класс используется для определения свойств уведомления (значок, звук и т. д.). Второй класс — системный сервис Android, который управляет всеми уведомлениями.

Первым делом нужно **получить ссылку** на NotificationManager через вызов getSystemService():

```
Context context = getApplicationContext();
NotificationManager Mgr =
(NotificationManager)getSystemService(Context.
  NOTIFICATION_SERVICE);
```

После этого создаем значок, текст уведомления и объект notify:

```
int icon = R.drawable.icon;
CharSequence cText = "Error!";
long t = System.currentTimeMillis();
Notification notify = new Notification(icon, cText, t);
```

Далее необходимо определить расширенный текст для уведомления:

```
CharSequence nTitle = "Error";
CharSequence nText = "Could not connect to server";
```

Создаем объект Intent:

```
Intent intent = new Intent(this, MyClass.class);
```

Далее делаем вызов:

```
PendingIntent cIntent =
PendingIntent.getActivity(this, 0, intent, 0);
```

Затем нужно указать расширенный текст и заголовок уведомления:

```
notify.setLatestEventInfo(context, nTitle, nText, cIntent);
```

Передаем notify в объект Mgr:

```
Mgr.notify(NOTIFY_ID, notify);
```

В Android предусмотрены диалоги следующих типов:

– AlertDialog — диалог с кнопками. Самый частый вариант применения диалога AlertDialog — это классический диалог вопроса с кнопками **Yes** и **No**. При запуске приложение отобразит диалог, при нажатии кнопки **No** не будет произведено никаких действий — будет вызван метод cancel(). Программная реализация кнопки **Yes** оставлена пустой.
– DatePickerDialog — диалог выбора даты.
– ProgressDialog — диалог отображает индикатор ProgressBar
– TimePickerDialog — диалог выбора времени.

Лекция 1.3. РАЗРАБОТКА МНОГОЭКРАННЫХ ПРИЛОЖЕНИЙ

На всех предыдущих лекциях создавались приложения, которые содержали только один экран (Activity). Но если использовать смартфоном с Android, то можно заметить, что экранов в приложении обычно больше. Если рассмотреть, например, почтовое приложение, то в нем есть следующие экраны: список аккаунтов, список писем, просмотр письма, создание письма, настройки и т.д. Рассмотрим особенности создания многоэкранных приложений.

Создадим приложение, в котором вызывается две активности.

При создании проекта по умолчанию создается Activity (рисунок 4.1).

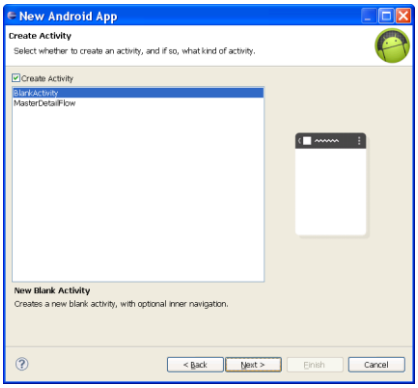

Рис. 4.1

От нас требуется только указать имя этого Activity – обычно здесь указывается MainActivity. Что при этом происходит? Мы знаем, что **создается одноименный класс** MainActivity.java – который отвечает за поведение Activity. Но, кроме этого, Activity «**регистрируется**» в системе с помощью манифест-файла - AndroidManifest.xml.

Нас интересует вкладка Application. Если его раскрыть, внутри видим Intent Filter с определенными параметрами (рисунок 4.2). Элемент android.intent.action.MAIN показывает системе, что Activity является основной и будет первой отображаться при запуске приложения. А android.intent.category.LAUNCHER означает, что приложение будет отображено в общем списке приложений Android.

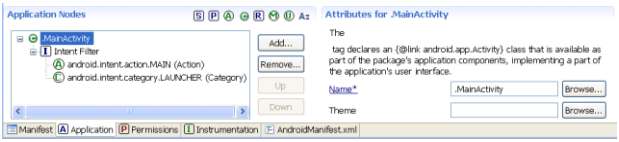

Рис. 4.2

Справа в поле Name написано MainActivity. Это имя класса, который отвечает за работу Activity (это же можно считать и именем Activity). Итак, если мы хотим создать еще одно Activity, надо создать класс и прописать Activity в AndroidManifest.xml. Чтобы создать класс, жмем правой кнопкой на package com.example.les_21 в папке проекта и выбираем New -> Class:

Рис. 4.3

В появившемся окне вводим имя класса – ActivityTwo, и суперкласс – android.app.Activity.

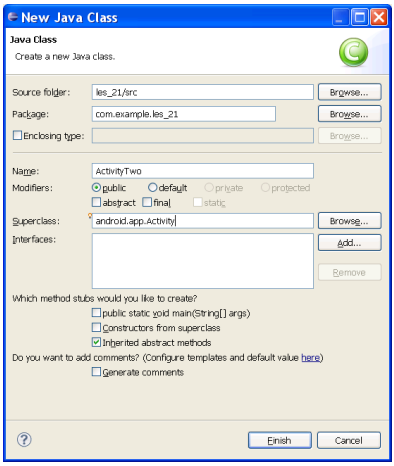

Рис. 4.4

Класс ActivityTwo создан. Он абсолютно пустой. Необходимо реализовать метод onCreate, который вызывается при создании Activity.

Не хватает вызова метода setContentView, который указал бы классу, чем заполнять экран. Этому методу на вход требуется layout-файл. Создадим его в папке layout, там же где и main.xml. Назовем файл second.xml:

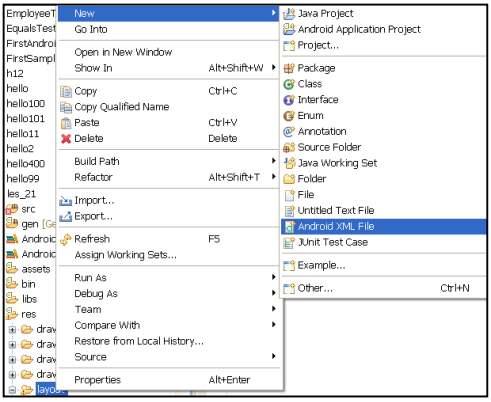

Рис. 4.5

Экран будет отображать TextView с текстом "This is Activity Two". Класс ActivityTwo готов, при создании он выведет на экран то, что настроено в layout-файле second.xml. Теперь необходимо прописать Activity в манифесте. Для этого надо открыть AndroidManifest.xml, вкладка Application. Нажать кнопку Add. Далее, в появившемся окне сверху выберите пункт «Create a new element at the top level …» (если есть выбор), а из списка выбираем Activity:

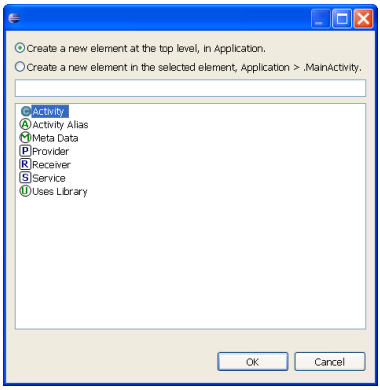

Рис. 4.6

Жмем OK, Activity создалось и появилось в списке. Осталось указать ему класс, который будет отвечать за его работу. Справа в поле Name можем написать вручную имя класса, который создавали – "ActivityTwo". А можем нажать Browse и выбрать его же из списка (надо будет немного подождать пока список сформируется). Кроме этого больше ничего указывать и заполнять не надо. Сохраним все.

Теперь в манифесте прописаны два Activity, и каждое ссылается на свой класс. Необходимо вернуться в MainActivity.java и завершить реализацию метода onClick (нажатие кнопки), а именно - прописать вызов ActivityTwo.

Рассмотрим код вызова Activity.

```
Intent intent = new Intent(this, ActivityTwo.class);
startActivity(intent);
```

Использован объект Intent. Рассмотрим данный объект более подробно.

В нашем случае Intent – это объект, в котором указывается, какое Activity необходимо вызвать. После чего этот Intent-объект передается методу startActivity, который находит соответствующее Activity и показывает его. При создании Intent использовался конструктор

Intent (Context packageContext, Class cls) с двумя параметрами.

Первый параметр – это Context. При программном создании View в конструкторах используется объект Context. Activity является подклассом Context, поэтому можно использовать ее – this. Вообще, Context – это объект, который предоставляет доступ к базовым функциям приложения таким как: доступ к ресурсам, к файловой системе, вызов Activiy и т.д.

Второй параметр – имя класса. При создании записи Activity в манифест-файле указывается имя класса. Поэтому если теперь указать тот же класс в Intent – то система, просмотрев манифест-файл обнаружит соответствие и покажет соответствующий Activity.

Вызов Activity с помощью такого Intent – это явный вызов. Т.е. с помощью класса явно указывается какое Activity необходимо отобразить. Это обычно используется внутри одного приложения. Схематично это можно изобразить так:

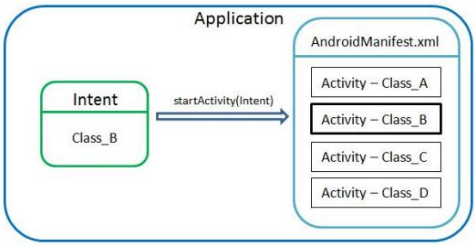

Рис. 4.7

Существует также неявный вызов Activity. Он отличается тем, что при создании Intent **используется не класс, а заполняются параметры action, data, category** определенными значениями. Комбинация этих значений определяет цель, которую необходимо достичь. Например: отправка письма, открытие гиперссылки, редактирование текста, просмотр картинки, звонок по определенному номеру и т.д. В свою очередь для Activity прописывается **Intent Filter** - это набор тех же параметров: action, data, category (но значения уже свои - зависят от того, что умеет делать Activity). И если параметры нашего Intent совпадают с условиями этого фильтра, то Activity вызывается. Но при этом поиск уже идет по всем Activity всех приложений в системе. Если находится несколько, то система предоставляет вам выбор, какой именно программой вы хотите воспользоваться. Схематично это можно изобразить так:

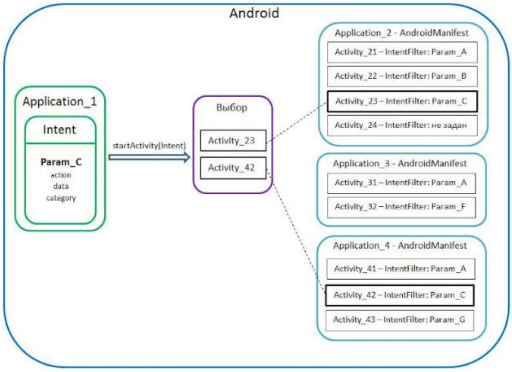

Рис. 4.8

В Application_1 создается Intent, заполняются параметры action, data, category. Для удобства, получившийся набор параметров назовем Param_C. С помощью startActivity этот Intent отправляется на поиски подходящей Activity, которая сможет выполнить то, что нам нужно

(т.е. то, что определено с помощью Param_C). В системе есть разные приложения, и в каждом из них несколько Activity. Для некоторых Activity определен Intent Filter (наборы Param_A, Param_B и т.д.), для некоторых нет. Метод startActivity сверяет набор параметров Intent и наборы параметров Intent Filter для каждой Activity. Если наборы совпадают (Param_C для обоих), то Activity считается подходящей.

Если в итоге нашлась только одна Activity – она и отображается. Если же нашлось несколько подходящих Activity, то пользователю выводится список, где он может сам выбрать какое приложение ему использовать.

Например, если в системе установлено несколько музыкальных плееров, и вы запускаете mp3, то система выведет вам список Activity, которые умеют играть музыку и попросит выбрать, какое из них использовать. А те Activity, которые умеют редактировать текст, показывать картинки, звонить и т.п. будут проигнорированы.

Если для Activity не задан Intent Filter (Activity_24 на схеме), то Intent с параметрами ему никак не подойдет, и оно тоже будет проигнорировано.

При работе приложения, пользователь создает новые Activity и закрывает старые, сворачивает приложение, снова открывает и т.д. Activity умеет обрабатывать все эти действия. Это необходимо, например, для освобождения ресурсов или сохранения данных. Созданное при работе приложения Activity может быть в одном из трех состояний:

- **Resumed** - Activity видно на экране, оно находится в фокусе, пользователь может с ним взаимодействовать. Это состояние также иногда называют Running.
- **Paused** - Activity не в фокусе, пользователь не может с ним взаимодействовать, но его видно (оно перекрыто другим Activity, которое занимает не весь экран или полупрозрачно).

– **Stopped** - Activity не видно (полностью перекрывается другим Activity), соответственно оно не в фокусе и пользователь не может с ним взаимодействовать.

Когда Activity переходит из одного состояния в другое, система вызывает различные методы, которые мы можем заполнять своим кодом. Схематично это можно изобразить так:

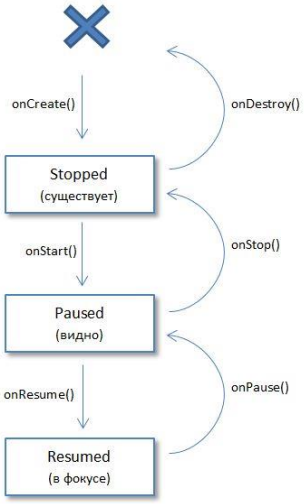

Рис. 4.9

Рассмотрим методы Activity, которые вызывает система:

– **onCreate()** – вызывается при первом создании Activity
– **onStart()** – вызывается перед тем, как Activity будет видно пользователю
– **onResume()** – вызывается перед тем как будет доступно для активности пользователя (взаимодействие)
– **onPause()** – вызывается перед тем, как будет показано другое Activity

– **onStop()** – вызывается когда Activity становится не видно пользователю
– **onDestroy()** – вызывается перед тем, как Activity будет уничтожено

Т.е. эти методы не вызывают смену состояния. Наоборот, смена состояния Activity является триггером, который вызывает эти методы. Тем самым нас уведомляют о смене, и мы можем реагировать соответственно.

Когда вы тестируете работу приложения, вы можете видеть логи работы. Они отображаются в окне LogCat. Чтобы отобразить окно откройте меню Window > Show View > Other … В появившемся окне выберите Android > LogCat

Добавим в DEBUG - логи с помощью метода Log.d. Метод требует на вход тэг и текст сообщения. Тэг – это метка, которая необходима для того, чтобы легче в списке системных логов найти требуемое сообщение.

Какие методы и в каком порядке выполняются при работе одного Activity, известно. Рассмотрим поведение при двух Activity.

Шаг1. Запускаем приложение. Появилось MainActivity. Вызываются три метода. Activity проходит через состояния Stopped, Paused и остается в состоянии Resumed:

Рис. 4.10

Шаг 2. Жмем кнопку «Go to Activity Two» на экране и появляется SecondActivity (рисунок 4.11). Вызов MainActivity.onPause означает, что MainActivity теряет фокус и переходит в состояние Paused. Затем создается (onCreate), отображается (onStart) и получает фокус (onResume) ActivityTwo. Затем перестает быть видно (onStop) MainActivity. Обратите внимание, что не вызывается onDestroy для MainActivity, а значит, оно не уничтожается. MainActivity остается в памяти, в состоянии Stopped. А SecondActivity – находится в состоянии Resumed. Его видно и оно в фокусе, с ним можно взаимодействовать.

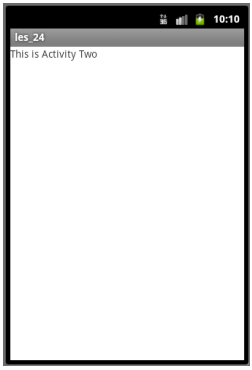

Рис. 4.11

Шаг 3. Жмем кнопку Назад (Back) на эмуляторе. Мы вернулись в MainActivity (рисунок 4.12). SecondActivity.onPause означает, что SecondActivity теряет фокус и переходит в состояние Paused. MainActivity теперь должна восстановиться из статуса Stopped. Метод onRestart вызывается перед методом onStart, если Activity не создается с нуля, а восстанавливается из состояния Stoped – так происходит здесь, MainActivity не было уничтожено системой, оно находилось в памяти. Поэтому вызывается MainActivity.onRestart. Далее вызываются методы MainActivity.onStart и MainActivity.onResume – значит MainActivity перешло в состояние Paused (отобразилось) и Resumed (получило фокус). Вызов методов onStop и onDestroy означает, что SecondActivity было переведено в статус Stopped (потеряло видимость) и было уничтожено.

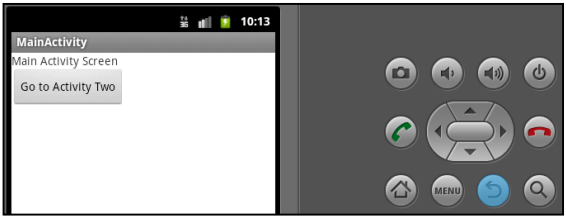

Рис. 4.12

Шаг 4. Жмем еще раз Назад и приложение закрылось. Логи показывают, что MainActivity перешло в состояние Paused, Stopped и было уничтожено.

Здесь для наглядности приведена схема шагов:

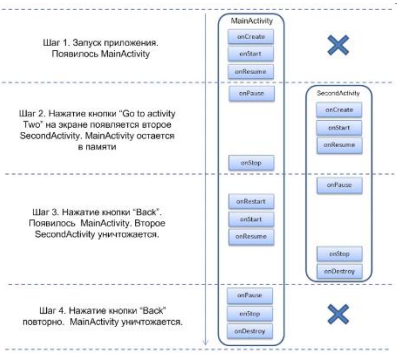

Рис. 4.13

Видно, что Activity не обязательно уничтожается, когда его не видно, а может оставаться в памяти. В связи с этим возникает вопрос: почему на шаге 2 MainActivity исчезло с экрана, но осталось в памяти и не было уничтожено? Ведь на шаге 3 было уничтожено SecondActivity после того, как оно пропало с экрана. А на шаге 4 было в итоге уничтожено и MainActivity. Почему шаг 2 стал исключением?

Разберемся куда помещается Activity, пока его не видно. И откуда оно извлекается при нажатии кнопки назад.

Известно, что приложение может содержать несколько Activity. И что Activity умеет вызывать Activity из других приложений с помощью Intent и Intent Filter. Если вы хотите отправить письмо из вашего приложения, вы вызываете Activity почтовой программы и передаете ей данные. Письмо уходит, и вы возвращаетесь в ваше приложение. Создается ощущение, что все это происходило в рамках одного приложения. Такая «бесшовность» достигается засчет того, что оба Activity (ваше и почтовое) были в одном **Task.**

Механизм организации Activity в Android очень схож по реализации с навигацией в браузере. Вы находитесь в одной вкладке (Task) и открываете страницы (Activity) переходя по ссылкам (Intent). В любой момент можете вернуться на предыдущую страницу, нажав кнопку Назад. Но кнопка Вперед отсутствует, т.к. страница, на которой была

нажата кнопка Назад, стирается из памяти. И надо снова нажимать ссылку, если хотим попасть на нее. Если вам надо открыть что-то новое, вы создаете новую вкладку и теперь уже в ней открываете страницы, переходите по ссылкам, возвращаетесь назад. В итоге у вас есть несколько вкладок. Большинство из них на заднем фоне, а одна (активная, с которой сейчас работаете) – на переднем.

Task – группа из нескольких Activity, с помощью которых пользователь выполняет определенную операцию. Обычно стартовая позиция для создания Task – это экран Домой (Home).

Находясь в Home вы вызываете какое-либо приложение из списка приложений или через ярлык. Создается Task. И Activity приложения (которое отмечено как MAIN в манифест-файле) помещается в этот Task как корневое. Task выходит на передний фон. Если же при вызове приложения, система обнаружила, что в фоне уже существует Task, соответствующий этому приложению, то она выведет его на передний план и создавать ничего не будет.

Когда Activity_A вызывает Activity_B, то Activity_B помещается на верх (в топ) Task и получает фокус. Activity_A остается в Task, но находится в состоянии Stopped (его не видно и оно не в фокусе). Далее, если пользователь жмет Back находясь в Activity_B, то Activity_B удаляется из Task и уничтожается. А Activity_A оказывается теперь на верху Task и получает фокус.

В каком порядке открывались (добавлялись в Task) Activity, в таком порядке они и содержатся в Task. Они никак специально не сортируются и не упорядочиваются внутри. Набор Activity в Task еще называют back stack.

Это состояние означает, что Activity не в фокусе, но оно видно, пусть и частично. Этого можно добиться, если присвоить диалоговый стиль для SecondActivity. Оно отобразится как всплывающее окно и под ним будет частично видно MainActivity – оно и будет в статусе Paused.

Для этого открываем AndroidManifest.xml, вкладка Application, находим там ActivityTwo и справа в поле Theme пишем такой текст: @android:style/Theme.Dialog.

Ранее был рассмотрен вызов Activity с помощью Intent и явного указания класса. Также кратко говорилось о другом способе вызова Activity – неявном. Он основан на том, что Activity вызывается не по имени, а по функционалу. Т.е. если необходимо выполнить определенные действия, то создается и настраивается соответствующий Intent и далее производится поиск тех Activity, которые смогли бы справиться с требуемой задачей.

Рассмотрим реализацию второго способа. Создадим приложение, которое будет отображать текущее время или дату. Реализация этого будет выполнена с помощью трех Activity:

- первое будет содержать две кнопки: Show time и Show date
- второе будет отображать время
- третье будет отображать дату

Нажатие на кнопку Show time будет вызывать второе Activity, а нажатие на кнопку Show date – третье Activity. При это реализация будет осуществлена посредством Intent Filter.

Для создания Intent используем конструктор: **Intent (String action).** Т.е. при создании заполняется атрибут объекта Intent, который называется action. Это обычная строковая константа. Action обычно указывает действие, которое мы хотим произвести. Например, есть следующие системные action-константы: ACTION_VIEW - просмотр, ACTION_EDIT – редактирование, ACTION_PICK – выбор из списка, ACTION_DIAL – сделать звонок.

Если действие производится с чем-либо, то в пару к action идет еще один Intent-атрибут – data. В нем можно указать какой-либо объект: пользователь в адресной книге, координаты на карте, номер телефона и т.п. Т.е. action указывает что делать, а data – с чем делать.

Здесь надо четко понимать следующее: action – это просто текст, поэтому он может быть произвольным. Но текст showtime – отражает то, что требуется сделать, он нагляднее и понятнее. А префикс lec8_1.intent.action. используется, чтобы не было коллизий. В системе может быть приложение, которое уже использует action showtime – наш с ним не должен пересекаться. Поэтому action называется lec8_1.intent.action.showtime.

Итак, был создан Intent с action и запущен в систему для поиска Activity. Чтобы Activity подошла, надо чтобы ее Intent Filter содержал атрибут action с тем же значением, что и action в Intent. Значит необходимо создать две Activity, настроить их Intent Filter и реализовать отображение времени и даты.

Также в Intent Filter надо создать Category и в поле name выбрать из списка android.intent.category.DEFAULT, без этого вызов startActivity(Intent) не найдет Activity.

Рассмотрим простое приложение. На первом экране вводится имя и фамилия, а второй экран будет эти данные отображать. Данные будут передаваться внутри Intent.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
     "http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:gravity="center_horizontal"
        android:text="Input your Name" >
    </TextView>
```

```
<TableLayout
     android:id="@+id/tableLayout1"
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:layout_margin="10dp"
     android:stretchColumns="1" >
     <TableRow
         android:id="@+id/tableRow1"
         android:layout_width="match_parent"
         android:layout_height="wrap_content" >
         <TextView
             android:id="@+id/textView1"
             android:layout_width="match_parent"
             android:layout_height="wrap_content"
             android:text="First Name" >
         </TextView>
         <EditText
             android:id="@+id/etFName"
             android:layout_width="match_parent"
             android:layout_height="wrap_content"
             android:layout_marginLeft="5dp" >
             <requestFocus>
             </requestFocus>
         </EditText>
     </TableRow>
    <TableRow
         android:id="@+id/tableRow2"
         android:layout_width="match_parent"
         android:layout_height="wrap_content" >
         <TextView
             android:id="@+id/textView2"
             android:layout_width="match_parent"
             android:layout_height="wrap_content"
             android:text="Last Name" >
```

```
            </TextView>
            <EditText
                android:id="@+id/etLName"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginLeft="5dp" >
            </EditText>
        </TableRow>
    </TableLayout>
    <Button
        android:id="@+id/btnSubmit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="OK" >
    </Button>
</LinearLayout>
```

В EditText будем вводится имя и фамилия, а кнопка OK будет вызывать другой экран и передавать ему эти данные.

В коде определяются поля ввода и кнопка. Кнопке присваиваем обработчик – Activity (this). Рассмотрим реализацию метода onClick. Intent создается с использованием класса, а не action. Это означает, что система просмотрит манифест файл приложения, и если найдет Activity с таким классом – отобразит его. ResultActivity пока не создан, поэтому код будет подчеркнут красным. Это не мешает нам сохранить файл. Чуть позже мы создадим это Activity и ошибка исчезнет.

Итак, Intent создан. Используется метод putExtra. Он имеет множество вариаций и аналогичен методу put для Map, т.е. добавляет к объекту пару. Первый параметр – это ключ (имя), второй - значение.

В Intent поместили два объекта с именами: fname и lname. fname содержит значение поля etFName, lname – значение поля etLName.

Остается только отправить укомплектованный Intent с помощью метода startActivity.

Теперь создадим второе Activity. Назовем его ResultActivity.

Создаем для него layout-файл result.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/tvView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="20dp"
        android:text="TextView"
        android:textSize="20sp" >
    </TextView>
</LinearLayout>
```

Находим TextView, затем получаем Intent и извлекаем из него String-объекты с именами fname и lname. Это те самые объекты, которые мы помещали в коде MainActivity.java. Формируем строку вывода в TextView с использованием полученных данных. Результат представлен на рисунке 5.1.

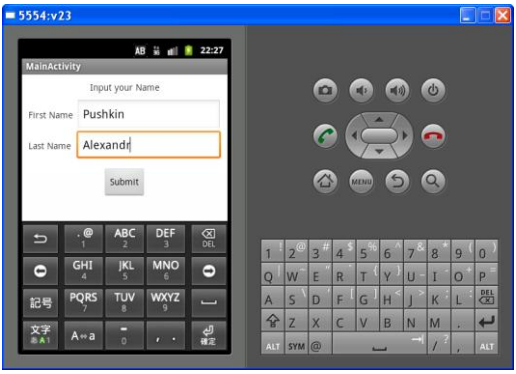

Рис. 5.1

Бывает необходимость вызвать Activity, выполнить на нем какое-либо действие и вернуть результат главному Activity. Например – при создании SMS. Вы жмете кнопку «добавить адресата», система показывает экран со списком из адресной книги, вы выбираете нужного вам абонента и возвращаетесь в экран создания SMS. Т.е. вы вызвали экран выбора абонента, а он вернул вашему экрану результат.

Известно, что Intent имеет атрибут action. С помощью этого атрибута обычно дается указание на выполнение какого-либо действия. Например, просмотр списка контактов или редактирование изображения. Но действие обычно совершается не просто так, а с чем-либо. Значит кроме указания действия, необходимо указывать на объект, с которым эти действия нужно произвести. Для этого Intent имеет атрибут data.

Один из способов присвоения значения этому атрибуту – метод setData (Uri data) у объекта Intent. На вход этому методу подается объект Uri.

Uri – это объект, который берет строку, разбирает ее на составляющие и хранит в себе эту информацию. Строка, должна быть не любая, а составлена в соответствии с документом RFC 2396. Uri имеет несколько методов, которые позволяют извлекать из обработанной строки отдельные элементы.

Рассмотрим такую строку - http адрес:

```
Uri uri = Uri.parse("http://developer.android.com/
  reference/android/net/Uri.html");
```

Смотрим, чего нам возвращают методы:

```
uri.getScheme(): http
uri.getSchemeSpecificPart():
//developer.android.com/reference/android/net/Uri.html
uri.getAuthority(): developer.android.com
uri.getHost(): developer.android.com
uri.getPath(): /reference/android/net/Uri.html
uri.getLastPathSegment(): Uri.html
```

Понятия Scheme, Authority, Host, Path и пр. имеются в RFC документа. Там можно найти их полное описание, понять что они означают и свериться с тем, что вернули методы Uri.

```
FTP
Uri uri = Uri.parse("ftp:// bob@google.com:80/
  data/files");
uri.getScheme(): ftp
uri.getSchemeSpecificPart()://bob@google.com:80/
  data/files
uri.getAuthority(): bob@google.com:80
uri.getHost(): google.com
uri.getPort(): 80
uri.getPath(): /data/files
uri.getLastPathSegment(): files
uri.getUserInfo(): bob
```

Координаты

```
Uri uri = Uri.parse("geo:55.754283,37.62002");
uri.getScheme(): geo
uri.getSchemeSpecificPart(): 55.754283,37.62002
```

Здесь уже получилось выделить только Scheme и SchemeSpecificPart.

Номер телефона

```
Uri uri = Uri.parse("tel:12345");
uri.getScheme(): tel
uri.getSchemeSpecificPart():12345
```

Аналогично, получилось выделить только две части из строки.

Контакт из адресной книги

```
Uri uri = Uri.parse("content://contacts/people/1");
uri.getScheme(): content
uri.getSchemeSpecificPart(): //contacts/people/1
uri.getAuthority(): contacts
uri.getPath(): /people/1
uri.getLastPathSegment(): 1
```

В этом примере Scheme равен content. Это особый тип данных – Content Provider. Он позволяет любой программе давать доступ к своим данным, а другим программам – читать и менять эти данные.

Примеры показывают, что Uri можно создать из абсолютно разных строк: http-адрес, ftp-адрес, координаты, номер телефона, контакт из адресной книги.

Тип содержимого можно определить по Scheme. И этот же Scheme можно настроить в Intent Filter и отсеивать Intent, только с нужным нам типом данных в Uri, например только http. Рассмотрим простой

пример, в котором будем формировать Intent с action и data, отправлять его и смотреть на результат. Попробуем просмотреть следующее: http-адрес, координаты на карте и открыть окно набора номера.

Чтобы посмотреть координаты на карте, необходимо приложение Google Maps.

В случае btnWeb использовался конструктор Intent (String action, Uri uri). Он создает Intent и на вход сразу принимает action и data. Мы используем стандартный системный action – ACTION_VIEW. Это константа в классе Intent – означает просмотр чего-либо. В качестве data подается объект Uri, созданный из веб-ссылки: http://developer.android.com. Этот Intent означает, что мы хотим посмотреть содержимое этой ссылки и ищем Activity, которая могла бы это выполнить.

В случае btnMap использовался конструктор Intent(). Он просто создает Intent. А в следующих строках происходит присвоение ему атрибутов action и data. action – снова ACTION_VIEW, а в качестве data создается Uri из пары координат - 55.754283,37.62002. Этот Intent означает, что мы хотим посмотреть на карте указанные координаты.

В случае btnCall используем конструктор Intent (String action). На вход ему сразу подается action, а data указывается позже. action в данном случае – ACTION_DIAL – открывает приложение телефон и набирает номер, указанный в data, но не начинает звонок. В data – помещаем Uri, созданный из номера телефона 12345.

Три этих способа приводят к одному результату - Intent с заполненными атрибутами action и data. Какой из них использовать - решать вам в зависимости от ситуации.

Т.к. нашему приложению понадобится интернет, чтобы открыть ссылку и посмотреть карту, надо чтобы на компьютере был интернет.

Также в файле манифеста приложения, на вкладке Permission добавьте элемент Uses Permission и справа в поле Name выберите

android.permission.INTERNET. Это даст приложению доступ в интернет.

Модуль 2. СОЗДАНИЕ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ НА ОСНОВЕ СЛОЖНЫХ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ

Лекция 2.1. ХРАНЕНИЕ ДАННЫХ В ANDROID-ПРИЛОЖЕНИЯХ (FILES AND PREFERENCES) ХРАНЕНИЕ ДАННЫХ В ANDROID-ПРИЛОЖЕНИЯХ (DATA BASE)

В Android есть несколько способов хранения данных:

– Preferences - в качестве аналогии можно привести Windows INI-файлы
– обычные файлы - внутренние и внешние (на SD карте)
– SQLite - база данных, таблицы

Рассмотрим Preferences. Значения сохраняются в виде пары: имя, значение.

Разработаем приложение. В нем будет поле для ввода текста и две кнопки – Save и Load. По нажатию на Save значение из поля будет сохраняться, по нажатию на Load – загружаться.

Код в MainActivity.java

```
public class MainActivity extends Activity
  implements OnClickListener {
  EditText etText;
  Button btnSave, btnLoad;
  SharedPreferences sPref;
  final String SAVED_TEXT = "saved_text";

  /** Called when the activity is first created. */
  @Override
```

```
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   etText = (EditText) findViewById(R.id.etText);
   btnSave = (Button) findViewById(R.id.btnSave);
   btnSave.setOnClickListener(this);
   btnLoad = (Button) findViewById(R.id.btnLoad);
   btnLoad.setOnClickListener(this);
   //loadText();
}

public void onClick(View v) {
   switch (v.getId()) {
      case R.id.btnSave:
      saveText();
      break;
      case R.id.btnLoad:
      loadText();
      break;
      default:
      break;
   }
}

void saveText() {
   sPref = getPreferences(MODE_PRIVATE);
   Editor ed = sPref.edit();
   ed.putString(SAVED_TEXT,
      etText.getText().toString());
   ed.commit();
   Toast.makeText(this, "Text saved",
      Toast.LENGTH_SHORT).show();
}
```

```
    void loadText() {
      sPref = getPreferences(MODE_PRIVATE);
      String savedText = sPref.getString(SAVED_TEXT,
        "");
      etText.setText(savedText);
      Toast.makeText(this, "Text loaded",
        Toast.LENGTH_SHORT).show();
    }
}
```

Рассмотрим методы, которые вызываются в onClick

saveText – сохранение данных. Сначала с помощью метода getPreferences получаем объект sPref класса SharedPreferences, который позволяет работать с данными (читать и записывать). Константа MODE_PRIVATE используется для настройки доступа и означает, что после сохранения, данные будут видны только этому приложению. Далее, чтобы редактировать данные, необходим объект Editor – он получается из sPref. В метод putString указывается наименование переменной – это константа SAVED_TEXT, и значение – содержимое поля etText. Чтобы данные сохранились, необходимо выполнить commit. И для наглядности выводится сообщение, что данные сохранены.

loadText – загрузка данных. Так же, как и saveText, с помощью метода getPreferences получаем объект sPref класса SharedPreferences. MODE_PRIVATE снова указывается, хотя и используется только при записи данных. Здесь Editor не используется т.к. сейчас необходимо только чтение данных. Чтение выполняется с помощью метода getString – в параметрах указываем константу - это имя, и значение по умолчанию (пустая строка). Далее записываем значение в поле ввода etText и выводим сообщение, что данные считаны.

Preferences-данные сохраняются в файлы и их можно просмотреть. Для этого в Eclipse откройте меню Window > Show View > Other и

выберите Android > File Explorer. Отобразилась файловая система эмулятора. Открываем data/data/com.example.lec9_1/shared_prefs и видим там файл MainActivity.xml. Если его выгрузить на ПК и открыть – можно увидеть следующее:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
 <map>
  <string name="saved_text">hello world</string>
  </map>
```

Обратите внимание, что в пути к файлу используется имя package.

Рассмотрим, откуда появилось наименование файла MainActivity.xml. Кроме метода getPreferences, который использовался, есть метод getSharedPreferences. Он выполняет абсолютно те же функции, но позволяет указать имя файла для хранения данных. Т.е., например, если бы в saveText использовался для получения SharedPreferences такой код:

```
sPref = getSharedPreferences("MyPref", MODE_PRIVATE);
```

То данные сохранились бы в файле MyPref.xml, а не в MainActivity.xml.

Теперь если рассмотреть исходный код метода getPreferences, то видим следующее:

```
public SharedPreferences getPreferences(int mode) {
  return getSharedPreferences(getLocalClassName(),
     mode);
}
```

Используется метод getSharedPreferences, а в качестве имени файла берется имя класса текущего Activity. Отсюда и появилось имя файла MainActivity.xml.

– используете getPreferences, если работаете с данными для текущего Activity и не хотите выдумывать имя файла.
– используете getSharedPreferences, если сохраняете, например, данные - общие для нескольких Activity и сами выбираете имя файла для сохранения.

Работа с файлами в Android не сильно отличается от таковой в Java.

При работе с файлами на SD используются стандартные java механизмы. При работе с внутренним хранилищем для удобства можно использовать методы-оболочки от Activity:

– openFileOutput – открыть файл на запись
– openFileInput – открыть файл на чтение
– deleteFile – удалить файл

Метод getFilesDir – возвращает объект File, соответствующий каталогу для файлов программы. Его можно использовать, чтобы работать напрямую, без методов-оболочек. Если у вас проверка SD-карты показывает, что карта недоступна, то убедитесь в свойствах AVD, что у вас для SDCard указан Size или File. Если указаны, то попробуйте перезапустить AVD.

Хранение данных с помощью SQLite

Это база данных с таблицами и запросами - все как в обычных БД.

В приложении, при подключении к БД мы указываем имя БД и версию. При этом могут возникнуть следующие ситуации:

1) БД не существует. Это может быть, например, в случае первичной установки программы. В этом случае приложение должно само создать БД и все таблицы в ней. И далее оно уже работает с только что созданной БД.

2) БД существует, но ее версия устарела. Это может быть в случае обновления программы. Например, новой версии программы нужны дополнительные поля в старых таблицах или новые таблицы. В этом случае приложение должно обновить существующие таблицы и создать новые, если это необходимо.

3) БД существует и ее версия актуальна. В этом случае приложение успешно подключается к БД и работает.

Для обработки описанных выше ситуаций необходимо создать класс, являющийся наследником для SQLiteOpenHelper.

Назовем его DBHelper. Этот класс предоставляет нам методы для создания или обновления БД в случаях ее отсутствия или устаревания.

onCreate - метод, который будет вызван, если БД, к которой мы хотим подключиться – не существует

onUpgrade - будет вызван в случае, если мы пытаемся подключиться к БД более новой версии, чем существующая

Разработаем простое приложение, которое будет хранить имя и возраст. Ввод данных будет осуществляться на экране приложения, а для отображения информации будут использоваться логи.

Рассмотрим приложение – справочник вузов. Возьмем 15 вузов и сохраним в БД их наименование, количество студентов и страну. Реализуем в приложении следующие функции:

- вывод всех записей
- вывод значения агрегатной функции (SUM, MIN, MAX, COUNT)
- вывод вузов с числом студентов, больше чем указано
- группировка вузов по стране
- вывод стран с числом студентов больше, чем указано
- сортировка вузов по наименованию, численности студентов и стране

Все данные выводятся в лог.

```
public void onClick(View v) {
  // подключаемся к базе
  db = dbHelper.getWritableDatabase();
  // данные с экрана
  String sFunc = etFunc.getText().toString();
  String sStudent = etStudent.getText().toString();
  String sCountryStudent =
    etCountryStudent.getText().toString();
  // переменные для query
  String[] columns = null;
  String selection = null;
  String[] selectionArgs = null;
  String groupBy = null;
  String having = null;
  String orderBy = null;
  // курсор
  Cursor c = null;
  // определяем нажатую кнопку
  switch (v.getId()) {
    // Все записи
    case R.id.btnAll:
      Log.d(LOG_TAG, "--- Все записи ---");
      c = db.query("studTable", null, null, null, null,
        null, null);
      break;
      // Функция
    case R.id.btnFunc:
      Log.d(LOG_TAG, "--- Функция " + sFunc + " ---");
      columns = new String[] { sFunc };
      c = db.query("studTable", columns, null, null,
        null, null, null);
      break;
    // Студентов больше, чем
    case R.id.btnStudent:
```

```
    Log.d(LOG_TAG, "--- Студентов больше " + sStudent
      + " ---");
    selection = "student > ?";
    selectionArgs = new String[] { sStudent };
    c = db.query("studTable", null, selection,
      selectionArgs, null,
    null, null);
    break;
// Студентов в стране
case R.id.btnGroup:
    Log.d(LOG_TAG, "--- Студентов в стране ---");
    columns = new String[] { "country", "sum(student)
      as student" };
    groupBy = "country";
    c = db.query("studTable", columns, null, null,
      groupBy, null, null);
    break;
// Студентов в стране больше чем
case R.id.btnHaving:
    Log.d(LOG_TAG, "--- Страны с числом студентов
      больше " + sCountryStudent
    + " ---");
    columns = new String[] { "country", "sum(student)
      as student" };
    groupBy = "country";
    having = "sum(student) > " + sCountryStudent;
    c = db.query("studTable", columns, null, null,
      groupBy, having,
    null);
    break;
// Сортировка
case R.id.btnSort:
    // сортировка по
    switch (rgSort.getCheckedRadioButtonId()) {
```

```
            // название вуза
            case (R.id.rName):
              Log.d(LOG_TAG, "--- Сортировка по названию
вуза ---");
              orderBy = "name";
              break;
            // число студентов
            case (R.id.rStudent):
              Log.d(LOG_TAG, "--- Сортировка по студентам
                ---");
              orderBy = "student";
              break;
            // страна
            case (R.id.rCountry):
              Log.d(LOG_TAG, "--- Сортировка по стране
                ---");
              orderBy = "country";
              break;
        }
        c = db.query("studTable", null, null, null, null,
          null, orderBy);
        break;
      }
      if (c != null) {
        if (c.moveToFirst()) {
          String str;
          do {
            str = "";
            for (String cn : c.getColumnNames()) {
              str = str.concat(cn + " = "
                + c.getString(c.getColumnIndex(cn)) + ";
                ");
              }
            Log.d(LOG_TAG, str);
```

```
        } while (c.moveToNext());
      }
    } else
      Log.d(LOG_TAG, "Cursor is null");
    dbHelper.close();
  }

  class DBHelper extends SQLiteOpenHelper {
    public DBHelper(Context context) {
      // конструктор суперкласса
      super(context, "myDB", null, 1);
    }
    public void onCreate(SQLiteDatabase db) {
      Log.d(LOG_TAG, "--- onCreate database ---");
      // создаем таблицу с полями
      db.execSQL("create table studTable "
        + "id integer primary key autoincrement," +
        "name text," + "student integer," + "country
        text" + ");");
      }

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

Три массива данных name, student, country. Это наименования вузов, численность студентов (в тысячах) и страны, к которым относятся вузы. По этим данным заполняется таблица.

В методе onCreate определяются и находятся экранные элементы, присваиваются обработчики, создается объект dbHelper для управления БД, подключение к базе данных и получение объекта db для работы с БД, проверка наличия записей в таблице, если нет ничего

– заполняется данными, закрывается соединение и эмулируется нажатие кнопки «Все записи» для того, чтобы сразу вывести в лог весь список.

В методе onClick производится подключение к базе, чтение данных с экранных полей в переменные, описание переменных, которые используются в методе query, и курсор. Далее определяется какая кнопка была нажата.

btnAll – вывод всех записей. Вызов метода query с именем таблицы и null для остальных параметров.

```
case R.id.btnAll:
  Log.d(LOG_TAG, "--- Все записи ---");
  c = db.query("studTable", null, null, null, null,
    null, null);
  break;
```

btnFunc – вывод значения агрегатной функции (или любого поля). Используется параметр columns, в который надо записать поля, которые необходимо получить из таблицы, т.е. то, что обычно перечисляется после слова SELECT в SQL-запросе. columns имеет тип String[] – массив строк. Создание массива из одного значения, которое считано с поля etFunc на экране.

btnStudent – вывод вузов с числом студентов больше введенного на экране количества. Используется selection для формирования условия. При этом используем один аргумент - ?. Значение аргумента задается в selectionArgs – это sStudent – содержимое поля etStudent.

btnGroup – группировка вузов по странам и вывод общего количества студентов. Используется columns для указания столбцов, которые необходимо получить – страна и общее число студентов. В groupBy указывается, что группировка будет по стране.

btnHaving – вывод стран с числом студентов больше указанного числа. Полностью аналогично случаю с группировкой, но добавляется условие в параметре having – общее число студентов в стране должна быть меньше sCountryStudent (значение etCountryStudent с экрана).

btnSort – сортировка стран. Определяем какой RadioButton включен и соответственно указываем в orderBy поле для сортировки данных.

В выше описанных случаях запускался query и получался объект с класса Cursor. Далее осуществлялась проверка, что он существует и в нем есть записи (moveToFirst). Если так, то запускается перебор записей в цикле do … while (c.moveToNext()). Для каждой записи перебираются названия полей (getColumnNames), получаем по каждому полю его номер и извлекаем данные методом getString. Формируется список полей и значений в переменную str, которая потом выводится в лог. После всего этого закрывается соединение.

Рассмотрим, как с помощью метода query выполнять запросы для связанных таблиц. Создадим простое приложение, которое будет делать запрос из двух таблиц и выводить результат в лог.

Таблицы будут people и position. В первую (people) запишем список людей, во вторую (position) – список должностей. И для каждого человека в people будет прописан id должности из position.

Экран использоваться не будет, поэтому activity_main.xml остается как есть.

Код MainActivity.java

```
public class MainActivity extends Activity {
  final String LOG_TAG = "myLogs";
  // данные для таблицы должностей
  int[] position_id = { 1, 2, 3, 4 };
  String[] position_name = { "Директор", "Программист",
    "Бухгалтер", "Охранник" };
  int[] position_salary = { 15000, 13000, 10000, 8000 };
  // данные для таблицы людей
```

```
String[] people_name = { "Иван", "Марья", "Петр",
  "Антон", "Даша", "Борис", "Костя", "Игорь" };
int[] people_posid = { 2, 3, 2, 2, 3, 1, 2, 4 };

/** Called when the activity is first created. */
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  // Подключаемся к БД
  DBHelper dbh = new DBHelper(this);
  SQLiteDatabase db = dbh.getWritableDatabase();
  // Описание курсора
  Cursor c;
  // выводим в лог данные по должностям
  Log.d(LOG_TAG, "--- Table position ---");
  c = db.query("position", null, null, null, null,
    null, null);
  logCursor(c);
  Log.d(LOG_TAG, "--- ---");
  // выводим в лог данные по людям
  Log.d(LOG_TAG, "--- Table people ---");
  c = db.query("people", null, null, null, null, null,
    null);
  logCursor(c);
  Log.d(LOG_TAG, "--- ---");
  // выводим результат объединения
  // используем rawQuery
  Log.d(LOG_TAG, "--- INNER JOIN with rawQuery---");
  String sqlQuery = "select PL.name as Name, PS.name
    as Position, salary as Salary "
    + "from people as PL "
    + "inner join position as PS "
    + "on PL.posid = PS.id "
    + "where salary > ?";
```

```
      c = db.rawQuery(sqlQuery, new String[] {"12000"});
      logCursor(c);
      Log.d(LOG_TAG, "--- ---");
      // выводим результат объединения
      // используем query
      Log.d(LOG_TAG, "--- INNER JOIN with query---");
      String table = "people as PL inner join position as
        PS on PL.posid = PS.id";
      String columns[] = { "PL.name as Name", "PS.name as
        Position", "salary as Salary" };
      String selection = "salary < ?";
      String[] selectionArgs = {"12000"};
      c = db.query(table, columns, selection,
         selectionArgs, null, null, null);
      logCursor(c);
      Log.d(LOG_TAG, "--- ---");
      // закрываем БД
      dbh.close();
      }

// вывод в лог данных из курсора
void logCursor(Cursor c) {
   if (c != null) {
      if (c.moveToFirst()) {
        String str;
        do {
          str = "";
          for (String cn : c.getColumnNames()) {
            str = str.concat(cn + " = " +
            c.getString(c.getColumnIndex(cn)) + "; ");
          }
          Log.d(LOG_TAG, str);
        } while (c.moveToNext());
      }
```

```
    } else
      Log.d(LOG_TAG, "Cursor is null");
  }

  // класс для работы с БД
  class DBHelper extends SQLiteOpenHelper {
    public DBHelper(Context context) {
      super(context, "myDB", null, 1);
    }

    public void onCreate(SQLiteDatabase db) {
      Log.d(LOG_TAG, "--- onCreate database ---");
      ContentValues cv = new ContentValues();
      // создаем таблицу должностей
      db.execSQL("create table position (“+ "id integer
        primary key,”+ "name text," + "salary integer”+
        ");");
      // заполняем ее
      for (int i = 0; i < position_id.length; i++) {
        cv.clear();
        cv.put("id", position_id[i]);
        cv.put("name", position_name[i]);
        cv.put("salary", position_salary[i]);
        db.insert("position", null, cv);
      }
      // создаем таблицу людей
      db.execSQL("create table people (“+ "id integer
        primary key autoincrement,"
        + "name text,”+ "posid integer”+ ");");
      // заполняем ее
      for (int i = 0; i < people_name.length; i++) {
        cv.clear();
        cv.put("name", people_name[i]);
        cv.put("posid", people_posid[i]);
```

```
            db.insert("people", null, cv);
        }
    }

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

Сначала идут несколько массивов с данными для таблиц. Обратите внимание, для должностей id указывается при заполнении таблиц. Это сделано для того, чтобы знать эти номера и использовать их в таблице людей для указания id должности.

В методе Activity onCreate создается объект для управления БД и подключение к БД. Далее используя query выводятся в лог данные из таблиц position и people.

Для вывода объединения таблиц используется rawQuery. Это несложный метод, который принимает на вход SQL-запрос и список аргументов для условия WHERE (если необходимо). В приложении сформирован запрос на объединение двух таблиц и вывода имени, должности и зарплаты человека. Условие выборки: ЗП должна быть больше 12000. Для формирования условия используются аргументы.

Далее снова выводится объединение таблиц, но используется обычный query. В table записываются все таблицы, их алиасы и условие JOIN. В columns – все нужные поля с использованием алиасов. В selection и selectionArgs записано условие выборки – ЗП меньше 12000.

Метод logCursor получает на вход Cursor и выводит в лог все содержимое.

Лекция 2.2. СОЗДАНИЕ И ИСПОЛЬЗОВАНИЕ СПИСКОВ В ANDROID-ПРИЛОЖЕНИЯХ

Прежде чем приступать к полноценному изучению списков рассмотрим класс LayoutInflater. Это пригодится при создании расширенных списков.

LayoutInflater – это класс, который из содержимого layout-файла создает View-элемент. Метод который это делает называется inflate. Есть несколько реализаций этого метода с различными параметрами. Но все они используют друг друга и результат их выполнения один – создание View объекта.

Рассмотрим реализацию

```
public View inflate (int resource, ViewGroup root,
   boolean attachToRoot)
```

На вход метод принимает три параметра:

resource - ID layout-файла, который будет использован для создания View. Например - R.layout.main

root – родительский ViewGroup-элемент для создаваемого View. LayoutParams от этого ViewGroup присваиваются создаваемому View.

attachToRoot – присоединять ли создаваемый View к root. Если true, то root становится родителем создаваемого View. Т.е. это равносильно команде root.addView(View). Если false – то создаваемый View просто получает LayoutParams от root, но его дочерним элементом не становится.

Объекты view — это основные модули для создания графического интерфейса пользователя на платформе Android. Класс view служит базовым для классов элементов управления, называемых виджетами, — текстовых полей, кнопок и т. д.

Объект view — структура, свойства которой сохраняют параметры компоновки и содержание для определенной прямоугольной области экрана. Как объект в интерфейсе пользователя объект view является точкой взаимодействия пользователя и программы.

Класс **viewGroup** представляет собой контейнер, который служит ядром для подклассов, называемых **компоновками** (layouts). Эти классы формируют расположение элементов пользовательского интерфейса на форме и содержат дочерние элементы View или ViewGroup.

Создадим **аналог** списка. В качестве данных будет использоваться информация о вузах из прошлой лекции: название вуза, страна и количество студентов. Т.е. каждый пункт нашего списка будет содержать три текстовых не редактируемых поля - name, country, student. Пункты будут размещены в виде вертикального списка.

Для реализации необходимы два layout-файла:

main.xml - основной экран для Activity, контейнер для пунктов списка

item.xml - экран с FrameLayout и тремя текстовыми полями в нем. Это будет пункт списка.

Приложение будет параллельно перебирать три массива данных, создавать для каждой тройки View-элемент из layout-файла item.xml, заполнять его данными и добавлять в вертикальный LinearLayout в main.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.
  com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="University list" >
    </TextView>
    <ScrollView
        android:id="@+id/scroll"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
        <LinearLayout
            android:id="@+id/linLayout"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical" >
        </LinearLayout>
    </ScrollView>
</LinearLayout>
```

ScrollView обеспечивает прокрутку списка, если все пункты не входят в экран. На экране находится LinearLayout, в который будут добавляться элементы.

Экран item.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.
  com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:layout_marginTop="10dp" >
    <TextView
        android:id="@+id/tvName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="top|center_horizontal"
        android:text="TextView"
```

```
            android:textSize="24sp" >
        </TextView>
        <TextView
            android:id="@+id/tvCountry"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="bottom|left"
            android:layout_marginLeft="5dp"
            android:text="TextView" >
        </TextView>
        <TextView
            android:id="@+id/tvStudent"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="bottom|right"
            android:layout_marginRight="5dp"
            android:text="TextView" >
        </TextView>
    </FrameLayout>
```

Запускается цикл по количествуву элементов в массивах данных. В каждой итерации создается View-элемент item из layout-файла item.xml. В данном случае item – это FrameLayout, который содержит три TextView. Производится их поиск в созданном item и заполняется данными из массивов.

В методе inflate в качестве root указан linLayout, чтобы получить от него LayoutParams и далее использовать для настройки ширины. Также для наглядности пункты списка подкрашены методом setBackgroundColor.

Обратите внимание - третий параметр inflate указан как false. Т.е. сразу не добавляется создаваемый View-элемент к linLayout, а делается это в конце кода методом addView. Если бы было указано true - то метод добавил бы item к linLayout и вернул бы linLayout, общий для

всех пунктов списка. Через linLayout заполнять TextView необходимым текстом было бы затруднительно. Поэтому сначала возвращается пункт item (FrameLayout), TextView заполняются данными и только потом помещается к остальным пунктам в linLayout методом addView.

Это еще не классический Android-список называемый List. Но этот пример значительно облегчит понимание списка т.к. принцип схож. Для построения List также необходимо будет предоставлять массив данных и layout-файл для пунктов.

В прошлом примере механизм построения списка был следующий: перебор массива данных, в каждой итерации создавался пункт списка, заполнялся данными из массива и помещался в список.

При создании ListView создавать пункты списка будет адаптер. Адаптеру нужны данные и layout-ресурс пункта списка. Далее адаптер присваивается списку ListView. Список при построении запрашивает у адаптера пункты, адаптер их создает (используя данные и layout) и возвращает списку. В итоге получается готовый список.

Существуют различные типы списков и адаптеров.

Адаптер – является своеобразным мостом между набором данных и объектом, использующим эти данные. Также адаптер отвечает за создание View-компонента для каждой единицы данных из набора.

В рассмотренных ранее примерах были:

– адаптер ArrayAdapter и объект для отображения данных ListView
– адаптер SimpleExpandableListAdapter и объект ExpandableListView.

Рассмотрим чем адаптеры отличаются друг от друга. На рисунке 7.1 приведена схема java-иерархии интерфейсов и классов адаптеров. В скобках указан тип: I – это интерфейс, AC – абстрактный класс, C – класс. Линии показывают наследование.

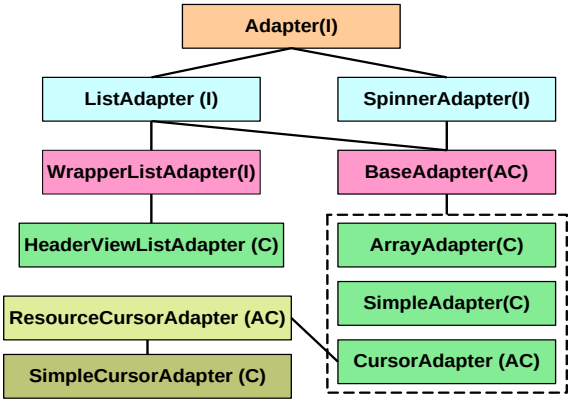

Рис. 7.1

Интерфейс Adapter. Описывает базовые методы, которые должны содержать адаптеры: getCount, getItem, getView и пр.

Интерфейс ListAdapter. Этот интерфейс должен быть реализован адаптером, который будет использован в ListView (метод setAdapter). Содержит описание методов для работы с разделителями (separator) списка.

Интерфейс SpinnerAdapter. Адаптеры, реализующие этот интерфейс, используются для построения Spinner (выпадающий список или drop-down). Содержит метод getDropDownView, который возвращает элемент выпадающего списка.

Интерфейс WrapperListAdapter. Адаптеры, наследующие этот интерфейс используются для работы с вложенными адаптерами. Содержит метод getWrappedAdapter, который позволяет извлечь из основного адаптера вложенный.

Класс HeaderViewListAdapter. Готовый адаптер для работы с Header и Footer. Внутри себя содержит еще один адаптер (ListAdapter), который можно извлечь с помощью выше рассмотренного метода getWrappedAdapter из интерфейса WrapperListAdapter.

Абстрактный класс BaseAdapter. Содержит свои методы и реализует методы интерфейсов, которые наследует, но не все. Своим наследникам оставляет на обязательную реализацию методы: getView, getItemId, getItem, getCount из ListAdapter. Т.е. если хотите создать свой адаптер – это класс вам подходит.

Класс ArrayAdapter<T>. Готовый адаптер, который уже использовался. Принимает на вход список или массив объектов, перебирает его и вставляет строковое значение в указанный TextView. Кроме наследуемых методов содержит методы по работе с коллекцией данных – add, insert, remove, sort, clear и метод setDropDownViewResource для задания layout-ресурса для отображения пунктов выпадающего списка.

Класс SimpleAdapter. Также готовый к использованию адаптер. Принимает на вход список Map-объектов, где каждый Map-объект – это список атрибутов. Кроме этого на вход принимает два массива – from[] и to[]. В to указываем id экранных элементов, а в from имена (key) из объектов Map, значения которых будут вставлены в соответствующие (из from) экранные элементы.

Т.е. SimpleAdapter – это расширенный ArrayAdapter. Если вы делаете ListView и у вас каждый пункт списка содержит не один TextView, а несколько, то вы используете SimpleAdapter. Кроме наследуемых методов SimpleAdapter содержит методы по наполнению View-элементов значениями из Map – setViewImage, setViewText, setViewBinder. Т.е. видим, что он умеет работать не только с текстом, но и с изображениями. Метод setViewBinder – позволяет вам написать свой парсер значений из Map в View-элементы и адаптер будет использовать его. Также содержит реализацию метода setDropDownViewResource.

Абстрактный класс CursorAdapter. Реализует абстрактные методы класса BaseAdapter, содержит свои методы по работе с курсором и

оставляет наследникам методы по созданию и наполнению View: newView, bindView.

Абстрактный класс ResourceCursorAdapter. Содержит методы по настройке используемых адаптером layout-ресурсов. Реализует метод newView из CursorAdapter.

Класс SimpleCursorAdapter. Готовый адаптер, похож, на SimpleAdapter. Только использует не набор объектов Map, а Cursor, т.е. набор строк с полями. Соответственно в массив from[] вы заносите наименования полей, значения которых хотите вытащить в соответствующие View из массива to.

Содержит метод convertToString, который возвращает строковое значение столбца, который задается методом setStringConversionColumn. Либо можно задать свой конвертер методом setCursorToStringConverter и адаптер будет использовать его при вызове convertToString. В этом конвертере вы уже сами реализуете, что он будет возвращать.

Итого существует 4 готовых адаптера: HeaderViewListAdapter, ArrayAdapter<T>, SimpleAdapter, SimpleCursorAdapter.

Если у вас есть массив строк то, можно использовать ArrayAdapter. Если работаете с БД и есть курсор, данные из которого надо вывести в список - используете SimpleCursorAdapter.

Если же эти адаптеры вам не подходят, есть набор абстрактных классов и интерфейсов, которые можно наследовать и реализовать в своих классах для создания своего адаптера. Да и готовые адаптеры всегда можно наследовать и сделать свою реализацию методов.

Известно, что SimpleAdapter умеет вставлять текст в TextView элементы и изображения в ImageView. Он использует для этого методы SetViewText и SetViewImage. Программист может создать свой адаптер на основе SimpleAdapter и реализовать эти методы под свои цели.

Эти методы предоставляют View объект и данные, следовательно, можно менять View в зависимости от данных. В качестве примера,

разработаем список, отражающий динамику какого-либо показателя по дням. Если динамика положительная –элементы будут зеленого цвета, если отрицательная –красного.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.
  com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <ListView
        android:id="@+id/lvSimple"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
    </ListView>
</LinearLayout>
```

На экране отображается только список.

Данные values упаковываются в коллекцию Map-объектов. Каждый Map будет состоять из трех атрибутов

ATTRIBUTE_NAME_TEXT - текст с номером дня

ATTRIBUTE_NAME_VALUE - значение динамики

ATTRIBUTE_NAME_IMAGE - id картинки для отображения, в зависимости от значения (положительное или отрицательное)

Затем заполняются массивы сопоставления данных (from) и View (to), создание адаптера, с использованием собственного класса MySimpleAdapter и формирование списка.

Рассмотрим реализацию MySimpleAdapter. Конструктор вызывает конструктор супер-класса.

Методы:

setViewText

Сначала выполняется метод супер-класса, который вставляет данные. Далее определяется идентификатор View, если это тот TextView, который будет отображать значения, то меняется цвет текста на красный или зеленый в зависимости от значения, которое он будет отображать.

setViewImage

Выполняется метод супер-класса, чтобы ImageView получил изображение, а дальше меняется его фон в зависимости от значения. Изображение содержит альфа-канал, поэтому фон будет виден. Проверку по id ImageView не выполняется, т.к. ImageView только один.

Адаптер SimpleAdapter при своей работе сопоставляет View-компоненты и значения из Map-объектов. Для расширения возможностей SimpleAdapter необходимо создать свой обработчик и присвоить его адаптеру.

Для этого используется метод setViewBinder (SimpleAdapter.ViewBinder viewBinder), который на вход требует объект SimpleAdapter.ViewBinder. Необходимо создать свой вариант этого биндера и реализовать в нем метод setViewValue(View view, Object data, String textRepresentation), в котором прописывается вся логика сопоставления данных и компонентов (биндинга). Метод возвращает значение boolean.

Алгоритм работы адаптера таков: он сначала проверяет, назначен ли ему сторонний биндер. Если он его находит, то выполняет его метод setViewValue. Если метод возвращает true, то адаптер считает, что обработка успешно завершена, если же false – то он выполняет биндинг в своем стандартном алгоритме.

Если адаптер не находит сторонний биндер, он также выполняет стандартный биндинг. Создадим пример, в котором будем заполнять значение ProgressBar и менять цвет LinearLayout. Это будет приложение-мониторинг, которое отображает уровень загрузки мощностей какой-то системы в разрезе дней. ProgressBar будет

показывать уровень загрузки, а весь пункт списка будет подкрашиваться цветом в зависимости от уровня загрузки.

Главный экран activity_main.xml:

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/
      android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical">
   <ListView
      android:id="@+id/lvSimple"
      android:layout_width="match_parent"
      android:layout_height="wrap_content">
   </ListView>
</LinearLayout>
```

Понадобится создать файл res/values/colors.xml, где перечислены требуемые цвета:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="Black">#9C9C9C</color>
   <color name="Red">#33FF0000</color>
   <color name="Orange">#33FFFF00</color>
   <color name="Green">#3300FF00</color>
   <color name="Black">#000000</color>
</resources>
```

Код MainActivity.java:

```
public class MainActivity extends Activity {
```

```
// имена атрибутов для Map
final String ATTRIBUTE_NAME_TEXT = "text";
final String ATTRIBUTE_NAME_PB = "pb";
final String ATTRIBUTE_NAME_LL = "ll";
ListView lvSimple;

/** Called when the activity is first created. */
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  // массив данных
  int load[] = { 10, 20, 30, 40, 50, 60, 70, 99 };
  // упаковываем данные в понятную
  // для адаптера структуру
  ArrayList<Map<String, Object>> data =
    new ArrayList<Map<String,
    Object>>(load.length);
  Map<String, Object> m;
  for (int i = 0; i < load.length; i++) {
    m = new HashMap<String, Object>();
    m.put(ATTRIBUTE_NAME_TEXT, "Process number " +
      (i+1) + ". Load: " + load[i] + "%");
    m.put(ATTRIBUTE_NAME_PB, load[i]);
    m.put(ATTRIBUTE_NAME_LL, load[i]);
    data.add(m);
  }
  // массив имен атрибутов, из которых будут
  // читаться данные
  String[] from = { ATTRIBUTE_NAME_TEXT,
    ATTRIBUTE_NAME_PB,ATTRIBUTE_NAME_LL };
  // массив ID View-компонентов, в которые будут
  // вставлять данные
  int[] to = { R.id.tvLoad, R.id.pbLoad, R.id.llLoad };
  // создаем адаптер
```

```
      SimpleAdapter sAdapter = new SimpleAdapter(this,
        data, R.layout.item,from, to);
      // Указываем адаптеру свой биндер
      sAdapter.setViewBinder(new MyViewBinder());
      lvSimple = (ListView) findViewById(R.id.lvSimple);
      // определяем список
      lvSimple.setAdapter(sAdapter);
      // присваиваем ему адаптер
    }
}

class MyViewBinder implements SimpleAdapter.ViewBinder {
    int red = getResources().getColor(R.color.Red);
    int orange =
      getResources().getColor(R.color.Orange);
    int green = getResources().getColor(R.color.Green);

    @Override
    public boolean setViewValue(View view, Object data,
      String textRepresentation) {
      int i = 0;
      switch (view.getId()) {
        // LinearLayout
        case R.id.llLoad:
          i = ((Integer) data).intValue();
          if (i < 40) view.setBackgroundColor(green);
          else
            if (i < 70)
              view.setBackgroundColor(orange);
            else
              view.setBackgroundColor(red);
          return true;
        case R.id.pbLoad:  // ProgressBar
          i = ((Integer) data).intValue();
```

```
            ((ProgressBar)view).setProgress(i);
            return true;
        }
        return false;
    }
}
```

Заполняем массив данных загрузки по 100-бальной шкале:

```
int load[] = { 10, 20, 30, 40, 50, 60, 70, 99 };
```

Формируем данные для адаптера: в TextView передаем краткую информацию (String), а в ProgressBar и LinearLayout – значение загрузки (int).

```
for (int i = 0; i < load.length; i++) {
  m = new HashMap<String, Object>();
  m.put(ATTRIBUTE_NAME_TEXT, "Process number " + (i+1)
    + ". Load: " + load[i] + "%");
  m.put(ATTRIBUTE_NAME_PB, load[i]);
  m.put(ATTRIBUTE_NAME_LL, load[i]);
  data.add(m);
}
```

Заполняем массивы сопоставления, создаем адаптер, говорим ему, чтобы использовал наш биндер, и настраиваем список.

```
  SimpleAdapter sAdapter = new SimpleAdapter(this, data,
R.layout.item, from, to);
  sAdapter.setViewBinder(new MyViewBinder());
```

Вложенный класс MyViewBinder – это реализация биндера. Необходимо реализовать метод **setViewValue**, который будет

использоваться адаптером для сопоставления данных из Map и View-компонентов. На вход ему идут:

view - View-компонент

data - данные для него

textRepresentation - текстовое представление данных (data.toString() или пустой String, но никогда не null)

Пример создания списка с возможностью удаления и добавления записей.

Добавление – кнопкой, а удаление – с помощью контекстного меню.

SimpleCursorAdapter отличается от SimpleAdapter тем, что в качестве данных используется не коллекция Map, а **Cursor** с данными из БД. И в массиве from, соответственно, указываются не ключи Map-атрибутов, а наименования полей (столбца) курсора. Значения из этих полей будут сопоставлены указанным View-компонентам из массива to.

Также немного отличается от SimpleAdapter стандартный биндинг и внешний ViewBinder. SimpleCursorAdapter умеет работать с TextView и ImageView компонентами и их производными, а Checkable-производные не поддерживает. При использовании ViewBinder, необходимо реализовать его метод

boolean setViewValue (View view, Cursor cursor, int columnIndex)

На вход он принимает View-компонент для биндинга, cursor с данными и номер столбца, из которого надо взять данные. Позиция курсора уже установлена в соответствии с позицией пункта списка.

Т.к. SimpleCursorAdapter – это адаптер для работы с данными из БД, то необходимо организовать БД.

Код по работе с БД рекомендуется выносить в отдельный класс DB.

В методе onCreate формируется коллекция Map-объектов, массивы сопоставления, создается адаптер и список, добавляется возможность контекстного меню для списка.

Метод onButtonClick – указан в main.xml в свойстве onClick кнопки. И теперь при нажатии на кнопку выполнится этот метод. Отдельный обработчик нажатия не нужен.

В этом методе создается новый Map, происходит добавление его к коллекции данных и сообщается, что данные изменились и надо обновить список.

Метод onCreateContextMenu – создание контекстного меню. Создается один пункт - для удаления записи.

В onContextItemSelected обрабатывается нажатие на пункт контекстного меню. При вызове контекстного меню объект, для которого оно было вызвано, передает в меню информацию о себе. Чтобы получить данные по пункту списка, для которого был совершен вызов контекстного меню, используется метод getMenuInfo.

Объект AdapterContextMenuInfo содержит данные о View, id и позиции пункта списка. Используется позиция для удаления соответствующего Map из коллекции. После этого сообщается, что данные изменились.

Методы-оболочки для работы с БД, предоставляют MainActivity только те возможности, которые ей нужны:

- open – установить соединение
- close – закрыть соединение
- getAllData – получить курсор со всеми данными из таблицы
- addRec – добавить запись
- delRec – удалить запись

Вложенный класс DBHelper используется для создания и управления БД. В методе onCreate создается таблица и заполняется сгенерированными данными.

В onCreate организовано подключение к БД, получен курсор.

Теперь при смене Lifecycle-состояний Activity, оно будет менять соответствующим образом состояния курсора. Затем настраивается

биндинг – формируются массивы, которые укажут адаптеру, как сопоставлять данные из курсора и View-компоненты.

Далее создается адаптер и настраивается список на его использование. В конце добавляется контекстное меню к списку.

В методе onButtonClick генерируется и добавляется запись в БД и обновляется курсор методом requery, чтобы получить свежие данные из БД.

При создании контекстного меню, в методе onCreateContextMenu, добавляется пункт для удаления.

В методе onContextItemSelected обрабатывается нажатие пункта контекстного меню. Чтобы получить данные по пункту списка, для которого был совершен вызов контекстного меню, используется метод getMenuInfo().

Объект AdapterContextMenuInfo() содержит данные о View, id и позиции пункта списка. Id равен значению поля _id для соответствующей записи в курсоре. Вызывается метод удаления записи и обновляется курсор.

В методе onDestroy подключение к БД закрывается. Это будет происходить при закрытии Activity.

Если список элементов получается большой, имеет смысл разбить его на группы для упрощения навигации. Для этих целей можно использовать ExpandableListView. Это список в виде двухуровневого дерева. Первый уровень – группа, а в ней второй – элемент из данной группы.

Чтобы построить такой список нужно как-то передать адаптеру данные по группам и элементам.

Каждая группа представляет из себя Map <String, ?>. Этот Map содержит атрибуты, которые нужны для каждой группы. Потом все Map (группы) собираются в List-коллекцию, например ArrayList. В итоге получаются упакованные в один объект группы.

Каждый элемент группы также представлен объектом Map<String, ?>. Все Map (элементы) собираются для каждой группы в отдельную коллекцию. Получается, каждой группе соответствует коллекция с элементами. Далее эти коллекции помещаются в общую коллекцию. Т.е. получается подобие двумерного массива. И в итоге пункты упакованы в один объект.

Сначала в классе описываются массивы данных – это названия групп и названия элементов для них. В качестве данных выбраны автомобили.

Затем описывается коллекция для групп, коллекции для элементов и Map для атрибутов.

```
ArrayList<Map<String, String>> groupData;
  ArrayList<Map<String, String>> childDataItem;
  ArrayList<ArrayList<Map<String, String>>> childData;
  Map<String, String> m;
```

В методе **onCreate** заполняется groupData. Это коллекция групп. Каждая группа представляет собой Map. В Map записываются необходимые атрибуты для каждой группы. В нашем случае, для каждой группы указывается всего один атрибут groupName - это название компании из массива groups.

```
for (String group : groups) {
   // заполняем список аттрибутов для каждой группы
   m = new HashMap<String, String>();
   m.put("groupName", group);
   groupData.add(m);
}
```

Адаптер обычно использует layout-ресурс для отображения пункта списка. В данном случае пунктами ListView являются

и группа и элемент. В layout-ресурсе могут быть какие-либо TextView. Мы можем заполнить их значениями из атрибутов элементов или групп, которые собраны в Map. Для этого необходимо указать сначала имена атрибутов, которые хотим использовать, а затем ID TextView-элементов, в которые хотим поместить значения этих атрибутов. Речь сейчас идет о текстовых атрибутах. (Хотя вообще атрибут вовсе не обязан быть класса String).

Для связки атрибутов и TextView-элементов используются два массива:

- **groupFrom** – список имен атрибутов, которые будут считаны - это groupName.
- **groupTo** – список ID View-элементов, в которые будут помещены считанные значения атрибутов.

Два этих массива сопоставляются по порядку элементов. В итоге, в layout-ресурсе группы найдется элемент с ID = android.R.id.text1 и в него запишется текст из атрибута groupName. Тем самым получается отображение имени группы (компании) в списке.

```
String groupFrom[] = new String[] {"groupName"};
        int groupTo[] = new int[] {android.R.id.text1};
```

Далее формируются коллекции элементов. Создается общая коллекция коллекций

```
childData = new ArrayList<ArrayList<Map<String,
  String>>>();
```

Затем создаются коллекции элементов каждой группы. Принцип тот же, что и с группами – создается Map и в него пишется атрибут carName со значением равным имени элемента.

```
childDataItem = new ArrayList<Map<String, String>>();
```

```
for (String car : carRenault) {
  m = new HashMap<String, String>();
    m.put("carName", car);
    childDataItem.add(m);
}
```

Коллекция элементов для каждой группы добавляется в общую коллекцию.

Необходимо создать адаптер **SimpleExpandableListAdapter** и присвоить его списку.

```
SimpleExpandableListAdapter adapter = new
  SimpleExpandableListAdapter(
            this,
            groupData,
            android.R.layout.simple_expandable
                 _list_item_1,
            groupFrom,
            groupTo,
            childData,
            android.R.layout.simple_list_item_1,
            childFrom,
            childTo);
```

На вход при создании адаптера поступают элементы:

this – контекст

groupData – коллекция групп

android.R.layout.simple_expandable_list_item_1 – layout-ресурс, который будет использован для отображения группы в списке.

groupFrom – массив имен атрибутов групп

groupTo – массив ID TextView из layout для групп

childData – коллекция коллекций элементов по группам

android.R.layout.simple_list_item_1 - layout-ресурс, который будет использован для отображения элемента в списке.

childFrom – массив имен атрибутов элементов

childTo - массив ID TextView из layout для элементов.

При использовании компонента ExpandableListView предоставлена возможность обрабатывать следующие события: **нажатие на группу, нажатие на элемент, сворачивание группы, разворачивание группы**.

Чтобы получить адаптер необходимо вызвать метод getAdapter.

У класса есть конструктор, через который передается объекту ссылка на context. Context необходим, чтобы создать адаптер. Адаптеру же в свою очередь context нужен, например, для доступа к LayoutInflater.

В конце класса находятся методы, которые возвращают названия групп и элементов из коллекций по номеру группы или номеру элемента. Для этого используются методы адаптера getGroup и getChild, приводим их к Map и извлекаем значение атрибута.

Благодаря классу AdapterHelper, код создания адаптера занимает две строчки: создание объекта и вызов метода getAdapter. Далее присваивается адаптер списку и добавляются обработчики:

1) `OnChildClickListener` - нажатие на элемент

Метод

```
public boolean onChildClick(ExpandableListView parent,
View v, int groupPosition, int childPosition, long id)
```

где

parent – ExpandableListView с которым осуществляется работа;

v – View элемента;

groupPosition – позиция группы в списке;

childPosition – позиция элемента в группе;

id – id элемента;

В лог выводится позиция и id. А в TextView сверху от списка выводится текст нажатого элемента и его группы, который получен с помощью методов AdapterHelper.

Метод должен вернуть boolean. Если он возвращает true – это значит, сообщается, что событие полностью обработано и оно не пойдет в дальнейшие обработчики (если они есть). Если возвращается false – значит событие идет дальше.

2) **OnGroupClickListener** – нажатие на группу

Метод

```
public boolean onGroupClick(ExpandableListView parent,
View v, int groupPosition, long id)
```

где

parent – ExpandableListView с которым работаем;

v – View элемента;

groupPosition – позиция группы в списке;

id – id группы;

Этот метод также должен вернуть boolean. Возвращается true, если позиция группы = 1, иначе - false.

3) **OnGroupCollapseListener –** сворачивание группы

Метод

```
onGroupCollapse(int groupPosition),
```

где

groupPosition – позиция группы, которую свернули.

4) **OnGroupExpandListener** – разворачивание группы

Метод

```
onGroupExpand(int groupPosition),
```

где

groupPosition – позиция группы, которую развернули.

Spinner – это выпадающий список, позволяющий выбрать одно значение. Он позволяет сэкономить место на экране.

Используем simple_spinner_item в качестве layout для отображения Spinner на экране. А методом **setDropDownViewResource** указываем какой layout использовать для прорисовки пунктов выпадающего списка.

Метод **setPrompt** устанавливает текст заголовка выпадающего списка, а **setSelection** – элемент, который мы хотим выделить. Оба метода, разумеется, необязательны.

Обработчик выбора элемента из списка присваивается методом **setOnItemSelectedListener.**

GridView – еще один из компонентов, использующих адаптеры. Он выводит элементы в виде сетки, матрицы или таблицы.

Атрибуты:

1. numColumns и columnWidth numColumns – кол-во столбцов в сетке. Если его не задавать, то столбец будет по умолчанию один. Это свойство также может иметь значение AUTO_FIT. В этом случае проверяется значение поля атрибута columnWidth (ширина столбца).

– если ширина столбца явно указана, то кол-во столбцов рассчитывается исходя из ширины, доступной GridView, и ширины столбцов.
– иначе, количество столбцов считается равным 2

2. HorizontalSpacing, verticalSpacing HorizontalSpacing, verticalSpacing - это горизонтальный и вертикальный отступы между ячейками. Пусть будет 5 и 10.

3. StretchMode - этот параметр определяет, как будет использовано свободное пространство, если оно есть. Используется в случае, когда вы указываете ширину столбца и количество ставите в режим AUTO_FIT. Изменим метод, добавим туда настройку stretch-параметра.

4. STRETCH_COLUMN_WIDTH – свободное пространство используется столбцами, это режим по умолчанию Столбцы растянуты по ширине. Она уже может не соответствовать той, что указана в setColumnWidth.

5. STRETCH_SPACING – свободное пространство равномерно распределяется между столбцами. Ширина столбцов неизменна. Увеличены интервалы между ними.

6. STRETCH_SPACING_UNIFORM – свободное пространство равномерно распределяется не только между столбцами, но и справа и слева. Ширина столбцов неизменна. Увеличены интервалы между ними и с боков.

Все эти параметры можно задавать не только программно, но и через атрибуты в layout-файлах.

Лекция 2.3. РАБОТА С ИСТОЧНИКАМИ ДАННЫХ (CONTENT PROVIDER) В ANDROID-ПРИЛОЖЕНИЯХ

Для устройств, размеры которых не позволяют использовать большие объемы памяти, как никогда актуально быстрое и эффективное сохранение и извлечение информации. У каждого приложения, работающего на платформе Android, есть доступ к легковесной реляционной базе данных SQLite. Программа может использовать все преимущества движка этой базы данных для безопасного и эффективного хранения информации. По умолчанию отдельные базы данных приложений изолированы друг от друга, то есть их содержимое может быть использовано только приложением, которое создало ту или иную базу. Однако Источники данных (Content Providers) обеспечивают возможность совместного использования баз данных приложений. Т.е. вы можете сконфигурировать собственный Content Provider, чтобы разрешить доступ к своим данным из других приложений, а также использовать Content Provider другого приложения для обращения к его хранилищу данных.

Android поддерживает три технологии передачи информации из приложения любому другому источнику: уведомления, классы переходов и Источники данных.

Уведомления— это стандартные средства, с помощью которых мобильные устройства что-либо сообщают пользователю. С помощью API можно вызывать звуковые сообщения, создавать вибрацию или отображать флеш-сообщения на экране устройства, а также менять статус значков уведомлений в строке состояния.

Классы переходов — это механизм передачи сообщений внутри приложений и между ними. С их помощью можно транслировать нужное действие (например, набор номера на телефоне или редактирование контакта) по всей системе в другие приложения, которые должны его обработать.

Источники данных – для открытия доступа к базам данных программы. Встроенные приложения, например, менеджер контактов, обеспечивают доступ к информации также через Источники данных, так что вы можете создавать программы, которые будут считывать или изменять эти данные.

Предоставляют интерфейс для публикации и потребления данных, основанный на простой адресной модели URI, используя схему content://. Этот механизм позволяет отделить логику приложения от данных, делая программы нечувствительными к источникам, из которых поступает информация, скрывая базовый источник данных.

Используются для получения результатов запросов, обновления или удаления существующих записей, а также для добавления новых. Любое приложение с соответствующими полномочиями может добавлять, удалять или изменять данные, принадлежащие другому приложению.

Многие стандартные базы данных доступны в качестве Источников данных и могут использоваться сторонними приложениями. Сюда входят телефонные контакты, хранилище информации и другие стандартные базы данных, речь о которых пойдет далее.

Публикуя собственные данные в виде Источников данных, вы получаете шанс (и даете его другим разработчикам) объединять и расширять их с помощью новых приложений.

Унифицированные идентификаторы содержимого (Content URI) в Android напоминают HTTP URI, но начинаются с **content** и строятся по следующему образцу:

content://authority-name/path-segment1/path-segment2/etc..

Пути URI должны быть уникальными и могут представляться двумя способами:

– content://com.google.provider.NotePad/notes - это запрос ко всем значениям определенного типа
– content://com.google.provider.NotePad/notes/23 - запрос к определенной строке

После content: в URI содержится унифицированный идентификатор источника, который используется для нахождения поставщика содержимого в соответствующем реестре. Часть URI com.google.provider.Notepad представляет собой источник - authority.

Фрагмент /notes/23 - это раздел пути (path section), специфичный для каждого отдельного поставщика содержимого. Фрагменты notes и 23 раздела пути называются сегментами пути (path segments). /Notes указывает на коллекцию записей (или каталог с записями), /23 - на определенную запись.

Создание нового источника данных:

– Необходимо наследовать абстрактный класс ContentProvider. Переопределить метод onCreate, чтобы создать (и инициализировать) базовый источник, который вы хотите опубликовать
– Предусмотрите публичное статическое свойство, которое возвращает полный URI для данного источника. URI Источника данных должен быть уникальным, поэтому лучше привязать его к имени вашего пакета.
– Используйте UriMatcher для поддержки двух видов доступа к вашему Источнику данных. Создайте и настройте UriMatcher, чтобы анализировать пути URI и распознавать их вид. В методе addUri UriMatcher подается комбинация: authority, path и константа. Причем, можно использовать спецсимволы: * - строка любых символов любой длины, # - строка цифр любой длины. На вход источнику данных будут поступать Uri, и будут отправляться в UriMatcher на проверку. Если Uri будет

подходить под комбинацию authority и path, ранее добавленных в addURI, то UriMatcher вернет константу из того же набора: authority, path, константа. Т.е. строка:

```
uriMatcher.addURI(AUTHORITY, CONTACT_TABLE,
  URI_CONTACTS);
```

означает, что в uriMatcher добавлена комбинация значений AUTHORITY, CONTACT_TABLE и URI_CONTACTS.
Строка:

```
uriMatcher.addURI(AUTHORITY, CONTACT_TABLE + "/#",
  URI_CONTACTS_ID);
```

означает, что в uriMatcher добавлена комбинация значений AUTHORITY, CONTACT_TABLE + "/#" и URI_CONTACTS_ID.
Где: # - это маска для строки из цифр.
А если к path добавляется число, это значит - мы будем работать с конкретной записью. Если uriMatcher проверяет Uri, состоящий из AUTHORITY и CONTACT_TABLE, он вернет значение URI_CONTACTS. А если Uri, состоит из AUTHORITY, CONTACT_TABLE и числа (ID), то он вернет URI_CONTACTS_ID. По этим константам можно определить – работать со всеми записями или какой-то конкретной.

– Назначьте имена каждому столбцу, доступному в вашем Источнике данных, чтобы упростить извлечение данных из результирующего Курсора.

Упростить доступ к запросам и транзакциям в Источнике данных, можно реализовав методы **delete, insert, update** и **query**.

Эти методы — интерфейс, используемый объектом **ContentResolver** для доступа к исходным данным. Они позволяют приложениям обмениваться данными в любой точке и не требуют разных интерфейсов для каждого источника данных. Объект UriMatcher задействуют для уточнения этих запросов и транзакций.

MIME - спецификация для кодирования информации и форматирования сообщений таким образом, чтобы их можно было пересылать по Интернету.

MIME типы работают в Android почти так же, как и в HTTP. Вы запрашиваете у источника данных тип MIME определенного поддерживаемого им URI, и он возвращает двухчастную последовательность символов, идентифицирующую тип MIME в соответствии с принятыми стандартами.

Обозначение MIME состоит из двух частей: типа и подтипа. Например: text/html, text/css, text/xml, application/pdf. Основные зарегистрированные типы содержимого: application, audio, image, message, model, multipart, text, video.

Обозначение vnd в типах MIME в Android означает, что данные типы и подтипы являются нестандартными. Тип и подтип должны быть уникальными для того типа содержимого, который они представляют. Для обеспечения уникальности в Android типы и подтипы разграничиваются при помощи нескольких компонентов.

Типы MIME всегда воспроизводятся источниками данных на основании соответствующих URI.

Обычно типы и подтипы относятся к определенному пространству имен в соответствии с вашими нуждами.

После создания источника данных, его нужно добавить в манифест приложения.

Используйте тег authorities, чтобы указать базовый путь. Когда система получит запрос на получение данных по Uri с нашим authority, она будет работать с нашим источником данных.

Name – имя класса источника данных

Authorities – уникальное имя, определяет источник данных

Далее Источник данных необходимо зарегистрировать:

```
<provider android:authorities="com.example.
  mycontentprovider.providers.AdressBook"
  android:name="MyContent">
</provider>
```

теперь, когда система получит запрос на получение данных по Uri с authority = com.example.mycontentprovider.providers.AdressBook, она будет работать с нашим источником данных.

Источник данных можно инсталлировать на AVD. На экране ничего не появится, т.к. нет Activity, а в консоли будут примерно такие строки:

Uploading P1011_ContentProvider.apk onto device 'emulator-5554' Installing P1011_ContentProvider.apk...Success! \P1011_ContentProvider\ bin\P1011_ContentProvider.apk installed on device Done!

Как можно заметить, практически ничего нового для нет. В основном идет работа с БД.

В начале идет описание констант для работы с БД. Будет всего одна таблица contacts с тремя полями: _id, name и email.

```
static final String DB_NAME = "mydb";
static final int DB_VERSION = 1;
// Таблица
static final String CONTACT_TABLE = "contacts";
// Поля
static final String CONTACT_ID = "_id";
static final String CONTACT_NAME = "name";
static final String CONTACT_EMAIL = "email";
```

```
// Скрипт создания таблицы
static final String DB_CREATE = "create table " +
  CONTACT_TABLE + "("
  + CONTACT_ID + " integer primary key autoincrement, "
  + CONTACT_NAME + " text, " + CONTACT_EMAIL + " text"
  + );
```

Далее идут константы AUTHORITY и CONTACT_PATH – это составные части Uri. Их этих двух констант и префикса content:// мы формируем общий Uri - CONTACT_CONTENT_URI. Т.к. здесь не указан никакой ID, этот Uri дает доступ ко всем контактам.

```
static final String AUTHORITY = "com.example.
  mycontentprovider.providers.AdressBook";
// path
static final String CONTACT_PATH = "contacts";
// Общий Uri
public static final Uri CONTACT_CONTENT_URI = Uri.parse
  ("content://"+ AUTHORITY + "/" + CONTACT_PATH);
```

Имя таблицы в БД, в данном случае, совпало с path в Uri. Это вовсе необязательно, они могут быть разными.

Далее описываем MIME-типы данных, предоставляемых провайдером. Один для набора данных, другой для конкретной записи.

```
static final String CONTACT_CONTENT_TYPE =
  "vnd.android.cursor.dir/vnd. " + AUTHORITY + "." +
  CONTACT_PATH;
// одна строка
static final String CONTACT_CONTENT_ITEM_TYPE =
  "vnd.android.cursor.item/vnd." +     AUTHORITY + "."
   + CONTACT_PATH;
```

Далее создаем и описываем UriMatcher и константы для него. uriMatcher определяет, какой Uri к нам пришел: общий или с ID. Если общий – то вернет URI_CONTACTS, если с ID – то вернет URI_CONTACTS_ID.

```
//// UriMatcher // общий Uri
static final int URI_CONTACTS = 1;
// Uri с указанным ID
static final int URI_CONTACTS_ID = 2;
// описание и создание UriMatcher
private static final UriMatcher uriMatcher;
static {
  uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
  uriMatcher.addURI(AUTHORITY, CONTACT_TABLE,
    URI_CONTACTS);
  uriMatcher.addURI(AUTHORITY, CONTACT_TABLE + "/#",
    URI_CONTACTS_ID);
}
```

Разбираем методы.

В OnCreate создаем DBHelper – обычный помощник для работы с БД.

В методе query мы получаем на вход Uri и набор параметров для выборки из БД: projection - столбцы, selection - условие, selectionArgs – аргументы для условия, sortOrder – сортировка.

Далее мы отдаем uri в метод match объекта uriMatcher. Он его разбирает, сверяет с теми комбинациями authority/path, которые мы давали ему в методах addURI и выдает константу из соответствующей комбинации. Если это URI_CONTACTS, значит нам пришел общий Uri и от провайдера хотят получить все его записи. В этом случае мы проверим, указана ли сортировка. Если нет, то поставим сортировку по имени. Операция с сортировкой совершенна необязательна. Если же

мы получили URI_CONTACTS_ID, то провайдер должен вернуть запись по конкретному ID. Для этого мы извлекаем ID из Uri методом getLastPathSegment и добавляем его в условие selection. Если uriMatcher не смог опознать Uri, то выдается IllegalArgumentException.

```
public Cursor query(Uri uri, String[] projection,
  String selection,String[] selectionArgs,
  String sortOrder) {
  // проверяем Uri
  switch (uriMatcher.match(uri)) {
    case URI_CONTACTS: // общий Uri
      //если сортировка не указана, ставим свою - по имени
      if (TextUtils.isEmpty(sortOrder)) {
        sortOrder = CONTACT_NAME + " ASC";
      }
      break;
    case URI_CONTACTS_ID: // Uri с ID
      String id = uri.getLastPathSegment();
      // добавляем ID к условию выборки
      if (TextUtils.isEmpty(selection)) {
        selection = CONTACT_ID + " = " + id;
      }
      else {
        selection = selection + " AND " + CONTACT_ID
          + " = " + id;
      }
      break;
    default: throw new
      IllegalArgumentException("Wrong URI: " + uri);
  }
```

Далее мы получаем БД и выполняем для нее метод query, получаем cursor. Регистрируем этот cursor, чтобы он получал уведомления, когда будут меняться данные, соответствующие общему Uri -

CONTACT_CONTENT_URI. При изменении какой-либо конкретной записи, уведомление также будет срабатывать. В конце возвращаем cursor.

```
    db = dbHelper.getWritableDatabase();
    Cursor cursor = db.query(CONTACT_TABLE, projection,
    selection,selectionArgs, null, null, sortOrder);
    // просим ContentResolver уведомлять этот курсор
    // об изменениях данных в CONTACT_CONTENT_URI
    cursor.setNotificationUri(getContext().
      getContentResolver(), CONTACT_CONTENT_URI);
    return cursor;
  }
```

В **insert** мы проверяем, что нам пришел наш общий Uri. Если да, то вставляем данные в таблицу, получаем ID. Этот ID добавляем к общему Uri и получаем Uri с ID (Это можно сделать и обычным сложением строк, но рекомендуется использовать метод withAppendedId.) Далее, уведомляем систему, что поменяли данные, соответствующие resultUri. Система посмотрит, не зарегистрировано ли слушателей на этот Uri. Увидит, что мы регистрировали курсор, и даст ему знать, что данные обновились. В конце мы возвращаем resultUri, соответствующий новой добавленной записи.

```
  public Uri insert(Uri uri, ContentValues values) {
    if (uriMatcher.match(uri) != URI_CONTACTS)
      throw new IllegalArgumentException
        ("Wrong URI: " + uri);
    db = dbHelper.getWritableDatabase();
    long rowID = db.insert(CONTACT_TABLE, null, values);
    Uri resultUri =
      ContentUris.withAppendedId(CONTACT_CONTENT_URI,
        rowID);
```

```
    // уведомляем ContentResolver, что данные по адресу
    // resultUri изменились
    getContext().getContentResolver().notifyChange
      (resultUri, null);
    return resultUri;
  }
```

В **delete** мы проверяем, какой Uri нам пришел. Если с ID, то фиксим selection – добавляем туда условие по ID. Выполняем удаление в БД, получаем кол-во удаленных записей. Уведомляем, что данные изменились. Возвращаем кол-во удаленных записей.

```
  public int delete(Uri uri, String selection,
    String[] selectionArgs) {
    switch (uriMatcher.match(uri)) {
      case URI_CONTACTS:
        break;
      case URI_CONTACTS_ID:
        String id = uri.getLastPathSegment();
        if (TextUtils.isEmpty(selection)) {
          selection = CONTACT_ID + " = " + id;
        } else {
          selection = selection + " AND " + CONTACT_ID
            + " = " + id;
        }
        break;
      default:
        throw new IllegalArgumentException
          ("Wrong URI: " + uri);
    }
    db = dbHelper.getWritableDatabase();
    int cnt = db.delete(CONTACT_TABLE, selection,
      selectionArgs);
    getContext().getContentResolver().notifyChange(uri,
```

```
      null);
    return cnt;
  }
```

В update мы проверяем, какой Uri нам пришел. Если с ID, то фиксим selection – добавляем туда условие по ID. Выполняем обновление в БД, получаем кол-во обновленных записей. Уведомляем, что данные изменились. Возвращаем кол-во обновленных записей.

Методе getType возвращает типы соответственно типу Uri – общий или с ID.

```
  public String getType(Uri uri) {
    switch (uriMatcher.match(uri)) {
      case URI_CONTACTS:
        return CONTACT_CONTENT_TYPE;
      case URI_CONTACTS_ID:
        return CONTACT_CONTENT_ITEM_TYPE;    }
      return null;
    }
```

Класс DBHelper помогает нам создать БД и наполнить ее первоначальными данными. Обновление здесь не реализуем.

```
  private class DBHelper extends SQLiteOpenHelper {
    public DBHelper(Context context) {
      super(context, DB_NAME, null, DB_VERSION);
    }
    public void onCreate(SQLiteDatabase db) {
      db.execSQL(DB_CREATE);
      ContentValues cv = new ContentValues();
      for (int i = 1; i <= 3; i++) {
        cv.put(CONTACT_NAME, "name " + i);
        cv.put(CONTACT_EMAIL, "email " + i);
```

```
            db.insert(CONTACT_TABLE, null, cv);
        }
    }
}
```

Каждый объект Context, принадлежащий приложению, включает в себя экземпляр класса ContentResolver, доступ к которому обеспечивает метод getContentResolver.

```
ContentResolver cr = getContentResolver();
```

ContentResolver применяет методы для изменения данных и выполнения запросов к Источникам данных. Каждый метод принимает URI, который указывает на то, с каким именно Источником данных нужно работать.

Путь URI Источника данных определяет его полномочия, описанные в соответствующем разделе манифеста. URI может быть произвольной строкой, так что большинство Источников данных имеют публичное свойство CONTENT_URI, чтобы предоставлять этот путь другим объектам. Источники данных, как правило, предоставляют два вида URI: один для запросов, требующих все данные сразу, а другой для запросов, которые возвращают одиночную строку. В последнем случае к CONTENT_URI в конце добавляется /<rowID>.

Запросы к Источникам данных весьма похожи на запросы к базам данных. Результат приходит в виде результирующего объекта Cursor.

Используя метод query, принадлежащий объекту ContentResolver, нужно передать ему:

– путь URI Источника данных, к которому требуется выполнить запрос;
– проекцию, содержащую список столбцов, которые хотите включить в результирующий набор данных;

– оператор WHERE, определяющий строки для возвращения (вы можете использовать маски «?», их заменят переданные значения для оператора SELECT);
– массив строк, выступающих в роли аргументов для оператора SELECT, которые должны заменить маски «?» в операторе WHERE;
– строку, описывающую порядок возвращения результирующих строк.

Для передачи транзакций к Источникам данных, используйте методы delete, update и insert, принадлежащие объекту ContentResolver.

Класс ContentResolver предлагает два метода добавления новых записей в Источник данных: **insert** и **bulkInsert**. Оба принимают в качестве параметра путь URI, описывающий тип элементов, но первый метод — одиночный объект ContentValues, а второй — массив этих объектов. Обычный метод **insert** вернет путь URI к только что добавленной записи, а **bulkInsert** вернет количество записей, которые были успешно добавлены.

Строки в Источнике данных обновляются с помощью метода **update**, принадлежащего ContentResolver. Данный метод принимает в качестве параметров путь URI, ссылающийся на нужный Источник данных, объект ContentValues, связывающий имена столбцов с обновленными данными, а также оператор WHERE, с помощью которого можно выбрать строки для обновления.

В процессе каждая строка, подходящая под описание оператора WHERE, обновляется, используя переданный объект ContentValues. При этом возвращается количество успешных обновлений.

Чтобы удалить одиночную запись, вызовите метод **delete** из объекта ContentResolver, передав ему путь URI той строки, которую хотите убрать. Кроме этого можно указать оператор WHERE для удаления нескольких строк.

В Android используются встроенные поставщики содержимого (пакет android.provider). На верхних уровнях иерархии располагаются базы данных, на нижних - таблицы. Так, Browser, CallLog, Contacts, MediaStore и Settings - это отдельные базы данных SQLite, инкапсулированные в форме поставщиков. Обычно такие базы данных SQLite имеют расширение DB и доступ к ним открыт только из специальных пакетов реализации (implerentation package). Любой доступ к базе данных из-за пределов этого пакета осуществляется через интерфейс поставщика содержимого. Вот неполный список: Browser, CallLog, Contacts, People, Phones, Photos, Groups, MediaStore, Audio, Albums, Artists, Genres, Playlists, Images, Thumbnails, Video, Settings

- Browser. Используется для чтения или изменения закладок, истории посещений или использования поиска в обозревателе.
- CallLog. Выводит или обновляет историю звонков (входящие, исходящие, пропущенные), открывает доступ к детальной информации, такой как номер звонившего и продолжительность разговора.
- ContactsContract. Используется для получения, изменения или хранения информации о контактах. Он заменяет старый класс Contacts.
- MediaStore. Предоставляет централизованный, управляемый доступ к файлам мультимедиа на вашем устройстве, включая аудио, видео и изображения. Вы можете хранить в нем собственные файлы мультимедиа и делать их общедоступными.
- Settings. Позволяет получить доступ к настройкам устройства с помощью этого Источника данных. Предоставляется возможность просматривать большинство системных настроек, а некоторые из них даже изменять. Класс android.provider.Settings содержит коллекцию действий для Намерений, которые могут

использоваться для открытия соответствующего экрана с настройками, чтобы пользователь мог их поменять.

– UserDictionary. Обеспечивает доступ к пользовательским наборам слов, добавленных в словарь для интеллектуального ввода текста.

MediaStore в Android можно назвать хранилищем аудио-, видеофайлов, а также изображений. Каждый раз, когда вы записываете на файловую систему файл мультимедиа, он также должен быть добавлен в MediaStore. Это откроет доступ к нему для других приложений, включая стандартный медиа проигрыватель.

Чтобы получить доступ к медиафайлам из MediaStore, нужно запрашивать их через Источники данных, как это описывалось ране. Класс MediaStore включает подклассы Audio, Video и Images, которые, в свою очередь, содержат подклассы, обеспечивающие доступ к именам столбцов и путям URI, ссылающимся на содержимое каждого Источника данных.

MediaStore упорядочивает файлы мультимедиа, хранящиеся на внутренних и внешних носителях устройства. Каждый из его подклассов предоставляет путь URI ко внутренним или внешним файлам, используя шаблоны вида:

– MediaStore.<mediatype>.Media.EXTERNAL_CONTENT_URI

– MediaStore.<mediatype>.Media.INTERNAL_CONTENT_URI

Доступ к управлению контактами понадобится, когда речь идет об устройствах для связи. Разработчики Android пошли верным путем и предоставляют всю информацию из базы данных контактов для любого приложения, имеющего полномочие **READ_CONTACTS**. Этот Источник данных включает новые возможности по управлению контактами в Android, предоставляя базу данных со всей информацией, касающейся контактов. Все это позволяет пользователям указывать несколько Источников своих контактных данных. Разработчики сами могут расширять информацию для каждого контакта или даже

создавать альтернативные Источники для контактов и сопутствующих данных.

Источник данных **ContactsContract** — это расширяемая база данных, содержащая всю информацию, связанную с контактами. Вместо использования одной строго определенной таблицы с контактной информацией **ContactsContract** оперирует трехуровневой моделью, отвечающей за хранение данных, связывание с контактом и объединение их для конкретного человека, используя следующие подклассы:

- **Data.** В исходной таблице каждая строка определяет набор личных данных (например, телефонные номера, адреса электронной почты и т. д.), разделенных типом MIME. Несмотря на предопределенный набор основных имен столбцов для каждого типа личных данных (доступных наряду с соответствующими MIME-типами, хранящимися в ContactsContract.CommonDataKinds), эта таблица может использоваться для хранения любого значения. То, какие именно данные находятся в конкретной строке, определяется с помощью типа MIME, указанного для нее. Универсальные столбцы способны хранить до 15 различных секций с данными разного типа. При добавлении новых данных в таблицу Data, нужно указать объект класса RawContacts для каждого связанного набора данных.
- **RawContacts.** Начиная с версии Android 2.0 пользователи могут вносить несколько контактных учетных записей (например, Gmail, Facebook и т. д.). Каждая строка в таблице RawContacts определяет учетную запись, с которой будут ассоциироваться данные из таблицы Data.
- **Contacts**. Эта таблица объединяет все строки из RawContacts, которые относятся к одному и тому же человеку.

– На деле таблица **Data** используется для добавления, удаления или изменения данных, относящихся к существующим учетным записям, **RawContacts** — для создания и управления самими учетными записями, **Contacts** и **Data** — для получения доступа к базе данных и извлечения информации о контактах.

Вы можете применять **ContentResolver** для запросов к любой из трех таблиц, описанных выше, используя статическую константу CONTENT_URI, предоставляемую каждым из этих классов. Все классы содержат статические свойства, описывающие имена столбцов исходной таблицы.

Чтобы получить доступ к любой контактной информации, необходимо добавить полномочие **READ_CONTACTS** к манифесту вашего приложения:

```
<uses-permission android:name="android.permission.
  READ_CONTACTS"/>
```

Источник данных **ContactsContract.Data** используется для хранения всей информации о контактах — адресов, телефонных номеров и адресов электронной почты. Это лучший вариант для поиска подобных данных. Для упрощения поисков Android предоставляет путь URI для запросов ContactsContract.Contacts.CONTENT_FILTER_URI. Добавьте к URI полное имя или его часть в качестве дополнительного сегмента пути. Для извлечения связанной с этим именем контактной информации найдите значение _ID в возвращенном объекте Cursor и используйте его для создания запроса к таблице Data.

Внутри таблицы **Data** содержание каждого столбца в строке зависит от типа MIME, указанного для этой строки. Поэтому каждый запрос к данной таблице должен фильтровать строки по типу MIME, чтобы явно извлекать данные. Подкласс **Contacts** предоставляет путь URI для поиска телефонного номера, чтобы помочь найти контакт, связанный с

номером телефона. Этот запрос оптимизирован для возвращения быстрых результатов об уведомлении о входящих звонках. Используйте путь ContactsContract.PhoneLookup.CONTENT_FILTER_URI, добавляя к нему номер, чтобы найти дополнительные участки пути.

В дополнение к статической контактной информации, таблица **ContactsContract.StatusUpdates** содержит обновления статусов в социальных сетях и данные о доступности в системах мгновенных сообщений. Используя эту таблицу, вы можете искать или изменять статус или сообщение о присутствии для любого контакта, который привязан к учетной записи в социальных сетях и/или системах мгновенных сообщений.

Кроме создания запросов к базе данных контактов можно использовать эти Источники данных для изменения, удаления или вставки записей о контактах, добавив полномочие **WRITE_CONTACTS** в манифест вашего приложения.

```
<uses-permission android:name="android.permission.
  WRITE_CONTACTS "/>
```

Благодаря принципу расширяемости ContactsContract вы можете добавлять произвольные строки в таблице Data к любой учетной записи, которая хранится в виде RawContacts. На деле это не лучший способ расширения сторонних учетных записей, так как нельзя синхронизировать новую информацию с удаленным сервером. Более удачное решение — создание собственного Адаптера для синхронизации, объединенного с другими сторонними данными об учетной записи. Существует возможность создания тип учетной записи контакта для собственного набора данных, добавив запись в Источник RawContacts.

Можно добавлять новые записи в Источник Data, которые будут привязаны к учетной записи контакта, созданной вами. После добавления ваши новые контактные данные объединятся с информацией, предоставляемой стандартными и сторонними Адаптерами, и станут доступны при запросах к Источнику ContactsContract, как описано в ранее.

Работа с мультимедиа

Мобильные устройства все больше и больше используются в качестве мультимедийных устройств. Многие устройства Android имеют встроенные камеры, микрофоны и громкоговорители, позволяя воспроизводить и записывать мультимедийные данные в различных форматах. Инструментарий Android SDK обеспечивает всестороннюю поддержку мультимедийных данных, позволяя разработчикам встраивать звуковые и визуальные элементы (изображения и видео) в приложения. Соответствующие интерфейсы API — это часть пакета **android.media.**

Эмулятор Android не позволяет записывать аудио или видео. Тестирование возможностей записи аудио и видео должно осуществляться на реальном устройстве Android. Кроме того, возможности записи отдельно взятого устройства могут зависеть от используемых аппаратных и программных компонент.

Существет два различных способа записи и воспроизведения аудироликов:

MediaPlayer/MediaRecorder — стандартный метод работы со звуком, который может храниться либо в файле, либо может быть представлен как потоковые данные. Для обработки аудио создается отдельный поток;

AudioTrack/AudioRecorder — прямой (raw) доступ к аудиоданным. Полезен при манипуляциях со звуком в памяти, записи звука в буфер при воспроизведении или в любых других случаях, не

требующих наличия потока или файла. Отдельный поток при обработке звука не создается.

Рассмотрим, как можно воспроизвести звук с помощью **MediaPlayer**. Первым делом нужно создать экземпляр **MediaPlayer**:

```
MediaPlayer media = new MediaPlayer();
```

Далее нужно указать источник звука, пусть это будет прямой (raw) ресурс:

```
media = MediaPlayer.create(this, R.raw.music1);
```

Можно загрузить звуковой файл:

```
media.setDataSource(путь);
media.prepare();
```

Запустить воспроизведение звука можно методом **start():**

```
media.start();
```

Метод **pauseMP()** приостанавливает воспроизведение, продолжить воспроизведение можно методом **startMP():**

```
media.pauseMP();
media.startMP();
```

Остановить воспроизведение и освободить ресурсы можно методами **stop()** и **release():**

```
media.stop();
media.release();
```

Для поддержки MediaPlayer нужно подключить следующий класс:

```
import android.media.MediaPlayer;
```

Прежде чем приступить к записи звука, нужно определиться с его источником (свойство **MediaRecorder.AudioSource**):

- MIC — встроенный микрофон;
- VOICE_UPLINK — исходящий голосовой поток при телефонном звонке (то, что вы говорите);
- VOICE_DOWNLINK — входящий голосовой поток при телефонном звонке (то, что говорит ваш собеседник);
- VOICE_CALL — запись телефонного звонка;
- CAMCORDER — микрофон, связанный с камерой, если таковой доступен;
- VOICE_RECOGNITION — микрофон, используемый для распознавания голоса, если таковой доступен.

После выбора источника звука нужно задать формат записываемого звука (свойство **MediaRecorder.OutputFormat**):

- THREE_GPP — формат 3GPP;
- MPEG_4 — формат MPEG4;
- AMR_NB — формат AMR_NB, лучше всего подходит для речи.

Последовательность действий для записи звука будет следующей. Сначала нужно создать экземпляр **MediaRecorder**:

```
MediaRecorder media = new MediaRecorder();
```

Затем нужно указать источник звука, например микрофон:

```
media.setAudioSource(MediaRecorder.AudioSource.MIC);
```

Третий шаг — установить результирующий формат и сжатие звука:

```
media.setOutputFormat(MediaRecorder.OutputFormat.AMR_NB);
media.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
```

Устанавливаем путь к файлу, в котором будут сохранены аудиоданные

```
media.setOutputFile(path);
```

Подготавливаем и запускаем запись:

```
media.prepare();
media.start();
```

Остановить запись можно методом **stop():**

```
media.stop();
```

Для поддержки **MediaRecorder** нужно подключить следующий пакет:

```
import android.media.MediaRecoder;
```

Возможностей **MediaPlayer/MediaRecorder** в большинстве случаев должно хватить. Но для манипуляции прямыми (raw) аудиоданными, полученными, например, непосредственно из микрофона, нужно использовать классы **AudioRecord** и **AudioTrack**. Первый класс используется для записи звука, второй — для воспроизведения.

Чтобы приложение могло записывать данные с помощью **AudioRecord**, нужно в файле манифеста объявить соответствующее разрешение

```
<uses-permission android:name="android.permission.
  RECORD_AUDIO"/>
```

Теперь рассмотрим процесс записи с помощью **AudioRecord**. Первым делом нужно создать экземпляр AudioRecord и установить источник звука:

```
short[] myAudio = new short(10000);
AudioRecord audioRecord = new AudioRecord(
  MediaRecorder.AudioSource.MIC, 11025,
  AudioFormat.CHANNEL_IN_MONO,
  AudioFormat.ENCODING_PCM_16BIT, 10000);
```

Наша конфигурация подходит для записи голоса со встроенного микрофона в буфер **myAudio**. Мы записываем 11 025 образцов в секунду, а размер буфера — 10 000 образцов, следовательно, длительность записи — менее секунды. Для более длительной записи нужно увеличить размер буфера.

Первый параметр — это источник звука. Вы можете использовать свойства **MediaRecorder.AudioSource** для указания источника звука.

Второй параметр (11 025) — это частота дискретизации звука (в Гц). Такая частота подходит только для записи голоса, для CD-качества необходима частота 44 100 Гц.

Мы записываем монозвук, для записи стерео измените третий параметр — его значение должно выглядеть как **AudioFormat.CHANNEL_IN_STEREO**.

Кодирование звука задается четвертым параметром. Здесь мы можем использовать либо 16-битное кодирование (что мы и делаем), либо 8-битное (**AudioFormat.ENCODING_PCM_8BIT**).

Последний параметр — это размер буфера в байтах, в который будет производиться запись. Другими словами — это общий размер выделенной памяти. Для правильного задания этого параметра лучше

было бы воспользоваться методом **getMinBufferSize()**, но для простоты примера мы указали просто значение в байтах.

Далее нужно начать запись:

audioRecord.startRecording();

Поскольку у нас прямой доступ к микрофону, то просто указать файл, в который нужно поместить прочитанный звук, нельзя. Нужно еще вручную считать этот звук с микрофона. Для этого используется метод **read()**:

```
audioRecord.read(myAudio, 0, 10000);
```

Остановить запись можно методом **stop():**

```
audioRecord.stop();
```

Для непосредственной манипуляции со звуком вам нужно подключить следующие пакеты:

```
import android.media.AudioFormat;
import android.media.AudioManager;
import android.media.AudioRecord;
import android.media.AudioTrack;
import android.media.AudioRecorder;
```

Теперь рассмотрим воспроизведение звука средствами **AudioTrack**. Конструктору объекта **AudioTrack** нужно передать:

- **тип потока** — AudioManager.STREAM_MUSIC (микрофон) или STREAM_VOICE_CALL (голосовой звонок). Другие варианты используются реже;
- **частоту в герцах** (Гц) — значения такие же, как и для записи звука;

- **конфигурацию канала** — AudioFormat.CHANNEL_OUT_STEREO или AudioFormat.CHANNEL_OUT_MONO. Можно также использовать значение CHANNEL_OUT_5POINT1 для звука 5.1;
- **тип кодирования звука** — значения такие же, как и для записи;
- **размер буфера в байтах**;
- **режим буфера** — AudioTrack.MODE_STATIC (подходит для небольших звуков, которые полностью помещаются в памяти) или AudioTrack.MODE_STREAM (для потокового звука).

Пример инициализации объекта **AudioTrack**:

```
AudioTrack audioTrack = new AudioTrack(
AudioManager.STREAM MUSIC, 11025,
AudioFormat.CHANNEL_OUT_MONO,
AudioFormat.ENCODING_PCM_16BIT, 4096,
AudioTrack.MODE_STREAM);
```

Далее нужно начать воспроизведение методом **play()**. Так как мы все делаем вручную, то должны вручную записать звуковые данные на устройство воспроизведения. Это делается методом **write()**. Первый параметр этого метода — буфер со звуковыми данными, а третий параметр — размер буфера. Второй параметр — смещение относительно начала буфера. Если смещение равно 0, то воспроизведение будет начато с начала буфера. Остановить воспроизведение можно методом **stop()**:

```
audioTrack.play();
audioTrack.write(myAudio, 0, 10000);
audioTrack.stop();
```

Для записи и воспроизведения видео используются классы MediaRecorder и MediaPlayer. Для записи видео нужно добавить в файл манифеста следующую строку:

```
<uses-permission android:name="android.permission.
  RECORD_VIDEO"/>
```

Источником видео (свойство MediaRecorder.VideoSource) может быть только встроенная камера — MediaRecorder. VideoSource.CAMERA.

Формат видео задается свойством OutputFormat:

THREE_GPP — формат 3GPP, в последнее время активно используется для записи мобильного видео;

MPEG_4 — популярный формат MPEG4.

Кодек видео можно выбрать с помощью свойства MediaRecorder.VideoEncoder:

– H264 — кодек H.264;
– H263 — кодек H.263;
– MPEG_4_SP — кодек MPEG4.

Рассмотрим последовательность действий по записи видео. Как обычно, сначала мы создаем объект класса MediaRecorder:

```
MediaRecorder VideoRecorder = new MediaRecorder();
```

Указываем источник видео (встроенная камера):

```
VideoRecorder.setVideoSource(MediaRecorder.
  VideoSource.CAMERA);
```

Третий шаг — установка формата файла и кодека:

```
VideoRecorder.setOutputFormat(MediaRecorder.
  OutputFormat.THREE_GPP);
```

```
VideoRecorder.setAudioEncoder(MediaRecorder.
  AudioEncoder.H263);
```

Путь к файлу, в котором будет сохранено видео, задается так:

```
VideoRecorder.setOutputFile(путь);
```

Осталось только начать запись:

```
VideoRecorder.prepare();
VideoRecorder.start();
```

Остановить запись можно методом **stop()**.

Рассмотрим пример программы VideoViewExample, которая воспроизводит видеофайлы.

Листинг программы:

```
package com.example.videoviewexample;
import android.app.Activity;
import android.net.Uri;
import android.os.Bundle;
import android.widget.MediaController;
import android.widget.VideoView;

public class VideoViewExample extends Activity {
   private VideoView mVideoView;
   @Override
   public void onCreate(Bundle icicle) {
     super.onCreate(icicle);
     setContentView(R.layout.
       activity_video_view_example);
     mVideoView = (VideoView)
       findViewById(R.id.surface_view);
```

```
      mVideoView.setVideoURI(Uri.parse
         ("android.resource://" + getPackageName()
           +"/"+R.raw.documentariesandyou));
      mVideoView.setMediaController(new
        MediaController(this));
      mVideoView.requestFocus();
    }
}
```

Листинг файла разметки страницы:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.
   com/apk/res/android"
     android:orientation="vertical"
     android:layout_width="match_parent"
     android:layout_height="match_parent">
    <VideoView
        android:id="@+id/surface_view"
        android:layout_width="320px"
        android:layout_height="240px"/>
</LinearLayout>
```

Результат работы программы:

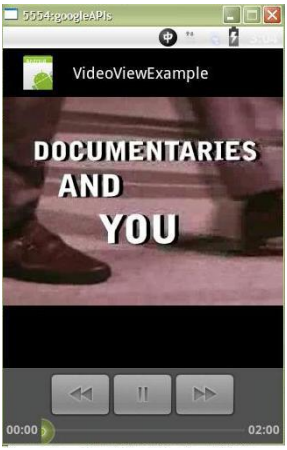
5554:googleAPIs
VideoViewExample
DOCUMENTARIES
AND
YOU
00:00
02:00

Рис. 9.1

Лекция 3.1. ВИДЖЕТЫ НА РАБОЧЕМ ЭКРАНЕ В ANDROID-ПРИЛОЖЕНИЯХ

Виджет – это небольшая программа, или часть программы, которая располагается на рабочем столе устройства и предназначена для отображения информации, управления оборудованием устройства и при этом может запускать другую программу, частью которой он является. Несколько виджетов предоставляет сам Android — это аналоговые часы, пульт управления проигрыванием музыки и еще один, показывающий картинки.

Множество разработчиков создали интересные виджеты с помощью которых можно отображать такие сведения, как загрузка процессора, состояние батареи, информацию о текущей погоде и прочем. Есть виджеты, с помощью которых можно быстро включить или выключить GPS, Wi-Fi, Bluetooth, динамики и управлять другим оборудованием Android устройства. Есть такие виджеты, как погодные информеры, которые отображают на экране информацию о текущей погоде и прогнозе погоды, и которые могут вызывать погодное приложение, частью которого они являются.

Виджеты могут иметь различный размер – от минимального размера 1x1, в стиле обычного ярлыка, до полноэкранного.

Таким образом, виджет это такая программа, которая «живет» на экране вашего планшета или телефона и позволяет вам управлять им, получать необходимую информацию и прочее.

В данной лекции будет изучено, как создавать свои собственные виджеты.

Чтобы создать простейший виджет нам понадобятся три детали:

1) Layout-файл. - В нем мы формируем внешний вид виджета. Все аналогично layout-файлам для Activity и фрагментов, только набор доступных компонентов здесь ограничен следующим списком:

– FrameLayout
– LinearLayout
– RelativeLayout
– GridLayout
– AnalogClock
– Button
– Chronometer
– ImageButton
– ImageView
– ProgressBar
– TextView
– ViewFlipper
– ListView
– GridView
– StackView
– AdapterViewFlipper

2) XML-файл с метаданными - В нем задаются различные характеристики виджета:

- layout-файл , чтобы виджет знал, как он будет выглядеть
- размер виджета, чтобы виджет знал, сколько места он должен занять на экране
- интервал обновления, чтобы система знала, как часто ей надо будет обновлять виджет

3) Класс, наследующий `AppWidgetProvider`.

В этом классе нам надо будет реализовать Lifecycle методы виджета.

onEnabled - вызывается системой при создании первого экземпляра виджета

onUpdate - вызывается при обновлении виджета. На вход, кроме контекста, метод получает объект AppWidgetManager и список ID экземпляров виджетов, которые обновляются. Именно этот метод

обычно содержит код, который обновляет содержимое виджета. Для этого нам нужен будет AppWidgetManager, который мы получаем на вход.

onDelete - вызывается при удалении каждого экземпляра виджета. На вход, кроме контекста, метод получает список ID экземпляров виджетов, которые удаляются.

onDisabled -вызывается при удалении последнего экземпляра виджета.

Чтобы запустить ваш новый виджет, перейдите в окно **Package Explorer**, щелкните правой кнопкой по проекту Widget и выберите команду **Run As ► Android Application**. К сожалению, вы не увидите никаких сообщений, которые бы могли как-то отобразить, как Eclipse компилирует и устанавливает виджет на ваш эмулятор или на устройство.

Чтобы увидеть новый виджет, откройте контекстное меню домашнего экрана: нажмите и удерживайте палец (или мышь) на домашнем экране. Появится меню с перечислением всех видов объектов, которые вы можете добавить.

Выберите **Widgets** из меню, затем выберите виджет, названный Widget. В итоге ваш виджет должен появиться на экране:

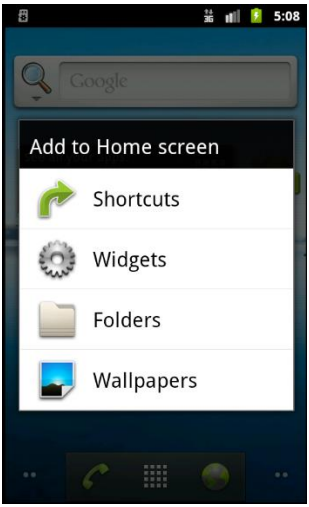

Рис. 12.1

Класс RemoteViews предназначен для описания и управления иерархиями Представлений, которые принадлежат к процессу другого приложения. Это позволяет изменять свойства или вызывать методы, принадлежащие Представлению, которое выступает частью другого приложения.

Например, Представления внутри виджетов работают в отдельном процессе (как правило, это домашний экран), поэтому RemoteViews может использоваться для изменения пользовательского интерфейса виджета из Приемника намерений, работающего внутри вашего приложения. Используя RemoteViews в сочетании с AppWidgetManager, вы получите возможность изменять внешний вид Представлений, которые поддерживаются фреймворком для создания виджетов. Среди прочего можно показывать и скрывать элементы, менять текст и изображения, добавлять реакцию на события нажатия.

На вход RemoteViews принимает имя пакета нашего приложения и ID layout-файла виджета. Теперь RemoteViews знает view-структуру нашего виджета. RemoteViews имеет несколько методов работы с view,

где мы указываем ID нужного нам view-компонента и значение, которое хотим передать. Из названия этих методов понятно, что они делают, например, setTextViewText. По названию понятно, что этот метод вставит текст в TextView. Мы вызываем его и передаем ID нашего TextView (из layout-файла виджета) и текст, который хотим в него поместить. Система потом найдет в виджете view с указанным ID и вызовет для него метод setText с указанным текстом.

Но таких явных методов немного. Они созданы просто для удобства и являются оболочками общих методов, которые позволяют вызвать любой метод view.

В названии общего метода содержится тип данных, которые вы хотите передать. А на вход методу кроме ID view и значения, необходимо будет указать имя метода.

Некоторые виджеты при размещении отображают конфигурационный экран, который позволяет настроить их. Например, у вас есть электронный счет на каком-либо сайте. И для этого сайта есть приложение-виджет. Чтобы виджет смог показать баланс именно вашего счета, он должен знать логин-пароль. Как вы понимаете, при разработке невозможно (если, конечно, вы не пишете виджет только для себя) зашить в код виджета нужный пароль и логин пользователя, поэтому эти данные надо у пользователя спросить.

Для этих целей и существует конфигурационный экран (конфигурационное Activity). Он предложит пользователю поля для ввода и сохранит куда-либо (БД, Preferences, …) введенные данные, а при обновлении виджета эти данные будут считаны и использованы для отображения актуальной информации.

Либо, например, мы хотим настроить внешний вид виджета при размещении.

Т.к. прямого доступа к view-компонентам виджета мы не имеем, то использовать, как обычно, обработчики нажатий не получится. Но RemoteViews, используемый для работы с view, позволяет настроить

реакцию view на нажатие. Для этого он использует PendingIntent. Т.е. мы можем на нажатие на виджет добавить вызов Activity, Service или BroadcastReceiver.

Создадим пример отражающий различные техники реагирования на нажатия. Виджет будет состоять из двух текстов и трех зон для нажатий.

Первый текст будет отображать время последнего обновления, а второй – количество нажатий на третью зону нажатия.

Первая зона будет по клику открывать конфигурационное Activity. Это пригодится в том случае, когда вы хотите дать пользователю возможность донастроить виджет после установки. Конфигурировать будем формат отображаемого в первой строке времени.

Вторая зона нажатия будет просто обновлять виджет, тем самым будет меняться время в первом тексте.

Каждое нажатие на третью зону будет увеличивать на единицу счетчик нажатий и обновлять виджет. Тем самым будет меняться второй текст, отображающий текущее значение счетчика:

Рис. 12.2

В третьей версии Андроид у виджетов появилась возможность работать с наборами данных типа списка или грида. Рассмотрим эту технологию на примере списка. В качестве view-компонента используется обычный ListView. Для межпроцессной работы с ним используется, как обычно в виджетах, RemoteViews. Но для заполнения нам придется создать два класса в дополнение к стандартному классу провайдера.

Первый – этот класс будет наполнять наш список значениями. Класс является реализацией интерфейса RemoteViewsService.RemoteViewsFactory, и его методы очень схожи с методами стандартного адаптера. Его обычно везде называют factory (адаптер).

Второй – класс сервиса, наследующий RemoteViewsService. В нем мы реализуем только один метод, который будет создавать и возвращать экземпляр (первого) класса, который будет заполнять список.

При создании и работе со списком в виджете необходимо понимать, как реализованы два момента: заполнение данными и реакция на нажатия.

При подготовке виджета в классе провайдера мы для списка присваиваем Intent, который содержит данные для вызова нашего второго класса-сервиса. Когда система хочет обновить данные в списке (в виджете) она достает этот интент, биндится к указанному сервису и берет у него адаптер. И этот адаптер уже используется для наполнения и формирования пунктов списка.

Теперь о реализации нажатий на пункты списка. В обычном виджете использовались PendingIntent. Здесь чуть по-другому. Для каждого пункта в списке НЕ создается свой отдельный PendingIntent. Вместо этого списку дается общий, шаблонный PendingIntent. А для каждого пункта списка мы указываем отдельный Intent с

extra-данными. Далее, при создании, каждому пункту списка система присваивает обработчик нажатия, который при срабатывании берет этот общий PendingIntent, добавляет к нему данные из персонального Intent, и отправляет по назначению сформированный таким образом PendingIntent. Т.е. в итоге по клику все равно срабатывает PendingIntent.

Для каждого пункта списка мы создаем Intent, помещаем в него позицию пункта и вызываем setOnClickFillIntent. Этот метод получает на вход ID View и Intent. Для View с полученным на вход ID он создает обработчик нажатия, который будет вызывать PendingIntent, который получается следующим образом: берется шаблонный PendingIntent, который был привязан к списку методом setPendingIntentTemplate (в классе провайдера) и к нему добавляется данные полученного на вход Intent-а. Т.е. получится PendingIntent, Intent которого будет содержать action = ACTION_ON_CLICK (это мы сделали еще в провайдере) и данные по позиции пункта списка. При нажатии на пункт списка, этот Intent попадет в onReceive нашего MyProvider и будет обработан

```
public RemoteViews getViewAt(int position) {
  RemoteViews rView = new
    RemoteViews(context.getPackageName(),
    R.layout.item);
  rView.setTextViewText(R.id.tvItemText,
    data.get(position));
  Intent clickIntent = new Intent();
  clickIntent.putExtra(MyProvider.ITEM_POSITION,
    position);
  rView.setOnClickFillInIntent(R.id.tvItemText,
    clickIntent);
  return rView;
}
```

onUpdate вызывается, когда поступает запрос на обновление виджетов. В нем мы перебираем ID, и для каждого вызываем метод updateWidget.

```
super.onUpdate(context, appWidgetManager,
  appWidgetIds);
for (int i : appWidgetIds)
  updateWidget(context, appWidgetManager, i);
```

updateWidget – здесь вызываем три метода для формирования виджета

```
setUpdateTV(rv, context, appWidgetId);
setList(rv, context, appWidgetId);
setListClick(rv, context, appWidgetId);
```

и затем метод updateAppWidget, чтобы применить все изменения к виджету.

```
appWidgetManager.updateAppWidget(appWidgetId, rv);
appWidgetManager.notifyAppWidgetViewDataChanged
  (appWidgetId,   R.id.lvList);
```

setUpdateTV – в этом методе работаем с TextView (который над списком). Ставим ему время в качестве текста

```
rv.setTextViewText(R.id.tvUpdate,
  sdf.format(new Date(System.currentTimeMillis())));
```

и добавляем обновление виджета по нажатию.

```
Intent updIntent = new
  Intent(context, MyProvider.class);
```

```
updIntent.setAction(AppWidgetManager.
  ACTION_APPWIDGET_UPDATE);
updIntent.putExtra(AppWidgetManager.
  EXTRA_APPWIDGET_IDS, new int[] { appWidgetId });
PendingIntent updPIntent = PendingIntent.
  getBroadcast(context, appWidgetId, updIntent, 0);
rv.setOnClickPendingIntent(R.id.tvUpdate,
  updPIntent);
```

setList – с помощью метода setRemoteAdapter указываем списку, что для получения адаптера ему надо будет обратиться к нашему сервису MyService.

Также в Intent мы помещаем ID виджета. Этот Intent будет передан в метод сервиса onGetViewFactory. Этот метод мы реализовывали, в нем мы создаем адаптер и передаем ему тот же Intent. А уже в адаптере достаем этот ID и используем (третья строка в списке).

Т.е. этот Intent пройдет через сервис и попадет в адаптер, поэтому если хотите что-то передать адаптеру, используйте этот Intent.

```
Intent adapter = new Intent(context, MyService.class);
adapter.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
  appWidgetId);
Uri data =
  Uri.parse(adapter.toUri(Intent.URI_INTENT_SCHEME));
adapter.setData(data);
rv.setRemoteAdapter(R.id.lvList, adapter);
```

setListClick – с помощью метода setPendingIntentTemplate устанавливаем шаблонный PendingIntent, который затем будет использоваться всеми пунктами списка. В нем мы указываем, что необходимо будет вызвать наш класс провайдера (он же BroadcastReceiver) с action = ACTION_ON_CLICK.

```
Intent listClickIntent = new
  Intent(context, MyProvider.class);
```

```
listClickIntent.setAction(ACTION_ON_CLICK);
PendingIntent listClickPIntent = PendingIntent.
  getBroadcast(context, 0, listClickIntent, 0);
rv.setPendingIntentTemplate(R.id.lvList,
  listClickPIntent);
```

onRecive - вызываем метод родителя, чтобы не нарушать работу провайдера. Далее проверяем, что action тот, что нам нужен - ACTION_ON_CLICK, получаем позицию нажатого пункта в списке и выводим сообщение на экран.

```
if (intent.getAction().equalsIgnoreCase
  (ACTION_ON_CLICK)) {
  int itemPos = intent.getIntExtra(ITEM_POSITION, -1);
  if (itemPos != -1) {
    Toast.makeText(context, "Clicked on item "
      + itemPos, Toast.LENGTH_SHORT).show();
  }
}
```

Для запуска приложения необходимо

- На главном экране нажмите на значок "Приложения".
- Нажмите Виджеты в верхней части экрана.
- Перетащите виджет на главный экран: нажмите и удерживайте значок виджета, перетащите его в нужное место и отпустите палец.

Видим время обновления виджета, время формирования данных в списке, хэш-код адаптера, ID виджета.

При нажатии на зеленую зону для обновления, время обновления виджета меняется. А при нажатии на какой-либо пункт выводится сообщение с его номером.

Лекция 3.2. СЛУЖБЫ

Службы в Android работают как фоновые процессы. Они не имеют пользовательского интерфейса, что делает их идеальными для задач, не требующих вмешательства пользователя. Служба может быть запущена и будет продолжать работать до тех пор, пока кто-нибудь не остановит ее или пока она не остановит сама себя.

Клиентские приложения устанавливают подключение к службам и используют это подключение для взаимодействия со службой. С одной и той же службой могут связываться множество клиентских приложений.

Служба имеет свои методы жизненного цикла:

– Void onCreate();
– Void onStart(Intent intent); onStartCommand
– Void onDestroy()

В полном жизненном цикле службы существует два вложенных цикла:

Полная жизнь службы – промежуток между временем вызова метода onCreate() и временем onDestroy().

Активная жизнь службы – начинается с вызова метода onStart(). Этому методу передается объект Intent, который передавали в StartService().

!!! Начиная с API 5 версии метод onStart обозначен как deprecated, вместо него используется onStartCommand

Из клиентского приложения службу можно запустить вызовом метода Context.startService(), остановить через вызов Context.stopService(). Служба может остановить сама себя, вызывая методы Service.stopSelf() или Service.stopSelfResult().

Можно установить подключение к работающей службе и использовать это подключение для взаимодействия со службой. Подключение устанавливают вызовом метода Context.bindService() и

закрывают Context.unbindService(). Если служба была остановлена, вызов метода bindService, может ее запустить.

Если служба разрешает другим приложениям связывать с собой, то привязка осуществляется с помощью дополнительных методов обратного вызова:

– Ibinder onBind(Intent intent);
– Boolean onUnbind(Intent intent);
– Void onRebind(Intent intent).

В метод обратного вызова onBind передают объект Intent, который был параметром bindService, а в метод обратного вызова onUnbind() – объект Intent, который передавали в метод unBindService.

Службу необходимо прописать в манифесте. При нажатии **Add**, появится список, в котором нужно выбрать Service:

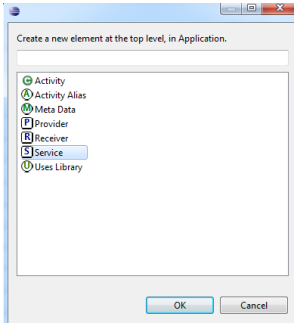

Рис. 15.1

Также в поле **Name** необходимо указать имя нашей службы MyService.

Основная форма приложения:

Рис. 15.2

Кроме метода stopSelf(), который вызывается внутри службы и останавливает ее, еще есть несколько вариаций:

– stopSelf(int startId)
– stopSelfResult(int startId)

startId – счетчик вызовов startService. Он является третьим входным параметром метода onStartCommand.

Именно этот startId и передается на вход методу stopSelf(startId). Т.е. этот метод дает системе понять, что конкретный вызов сервиса был успешно обработан.

Сервис останавливается, когда последний полученный (а не последний обработанный) вызов выполняет метод stopSelf(startId).

Метод stopSelfResult аналогичен методу stopSelf(int startId), но с тем отличием, что он возвращает значение типа boolean – остановлен сервис или нет.

– В **onCreate** создаем некий объект someRes. Этот объект будет использоваться сервисом в обработках вызовов.
– Executors.newFixedThreadPool(1) – эта строка дает нам объект, который будет получать от нас **задачи** (Runnable) и запускать их по очереди в **одном** потоке (на вход ему мы передаем

значение **1**). Он сделает за нас всю работу по управлению потоками.

- В **onStartCommand** мы читаем из intent параметр time. Создаем Runnable-объект **MyRun**, передаем ему time и startId и отдаем этот объект **экзекутору**, который его запустит в отдельном потоке.
- **MyRun** – Runnable-объект. Он и будет обрабатывать входящие вызовы сервиса. В конструкторе он получает **time** и **startId**. Параметр **time** будет использован для кол-ва секунд **паузы** (т.е. эмуляции работы). А **startId** будет использован в методе **stopSelf(startId)**, который даст сервису понять, что вызов под номером **strartId** обработан. В лог выводим инфу о создании, старте и завершении работы. Также здесь используем объект someRes, в лог просто выводим его класс. Если же объект = null, то ловим эту ошибку и выводим ее в лог.

!Не забудьте добавить в манифест наш сервис

PendingIntent является ещё одним механизмом, к помощью которого в Activity можно получать результаты работы службы.

Схема работы:

1) Создать в **Activity PendingIntent** с помощью метода **createPendingResult**
2) Вложить **PendingIntent** в обычный **Intent**, который используется для старта службы
3) Вызвать **startService**
4) В службе извлечь **PendingIntent** из полученного в методе **onStartCommand** объекта **Intent**
5) При необходимости передачи результатов работы из службы в **Activity**, вызвать метод **send** для объекта **PendingIntent**
6) Перехватить результаты из службы в **Activity** в методе **onActivityResult**

BroadcastReceiver позволяет прослушивать все Intent объекты возникающие в системе и извлекать только совпадающие с **IntentFilter**.

Схема работы:

1) В **Activity** создать **BroadcastReceiver** и **IntentFilter**, настроенный на определенный **Action**.
2) Зарегистрировать эту пару
3) При необходимости передачи результатов работы из службы в **Activity**, создать Intent объект, заполнить его данными и отправить.
4) Перехватить результаты из службы в **Activity** в методе **onReceive** объекта **BroadcastReceiver**

Существует синхронный способ взаимодействия с сервисом. Он достигается с помощью биндинга (binding). При этом, подключившись к службе, появляется возможность взаимодействовать с ней путем обычного вызова методов с передачей данных и получением результатов.

В качестве примера рассмотрим приложение и службу. Приложение способно подключиться к службе, настроить параметры службы и отключиться от службы. Служба в свою очередь будет выполнять задачи и выводить сообщения в лог.

```
import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.util.Log;

public class MyService extends Service {
  final String LOG_TAG = "myLogs";
  public void onCreate() {
    super.onCreate();
```

```
    Log.d(LOG_TAG, "MyService onCreate");
  }
  public IBinder onBind(Intent intent) {
    Log.d(LOG_TAG, "MyService onBind");
    return new Binder();
  }
  public void onRebind(Intent intent) {
    super.onRebind(intent);
    Log.d(LOG_TAG, "MyService onRebind");
  }
  public boolean onUnbind(Intent intent) {
    Log.d(LOG_TAG, "MyService onUnbind");
    return super.onUnbind(intent);
  }
  public void onDestroy() {
    super.onDestroy();
    Log.d(LOG_TAG, "MyService onDestroy");
  }
}
public class MyService extends Service {
  final String LOG_TAG = "myLogs";
  MyBinder binder = new MyBinder();
  Timer timer;
  TimerTask tTask;
  long interval = 1000;
  public void onCreate() {
    super.onCreate();
    Log.d(LOG_TAG, "MyService onCreate");
    timer = new Timer();
    schedule();
  }
  void schedule() {
    if (tTask != null) tTask.cancel();
    if (interval > 0) {
```

```
      tTask = new TimerTask() {
        public void run() {
          Log.d(LOG_TAG, "run");
        }
      };
      timer.schedule(tTask, 1000, interval);
    }
  }
long upInterval(long gap) {
    interval = interval + gap;
    schedule();
    return interval;
  }
  long downInterval(long gap) {
    interval = interval - gap;
    if (interval < 0) interval = 0;
    schedule();
    return interval;
  }
  public IBinder onBind(Intent arg0) {
    Log.d(LOG_TAG, "MyService onBind");
    return binder;
  }
  class MyBinder extends Binder {
    MyService getService() {
      return MyService.this;
    }
  }
}
```

Здесь мы используем таймер – **Timer**. Он позволяет повторять какое-либо действие через заданный промежуток времени. **TimerTask** – это задача, которую **Timer** будет периодически выполнять. В методе **run** описаны действия этой задачи. И далее для объекта **Timer** вызываем метод **schedule**, в который передаем задачу **TimerTask**,

время через которое начнется выполнение, и период повтора. Чтобы отменить выполнение задачи, необходимо вызвать метод **cancel** для **TimerTask**. Отмененную задачу нельзя больше запланировать, и если снова надо ее включить – необходимо создать новый экземпляр **TimerTask** и отправить его таймеру.

В методе **onCreate** мы создаем таймер и выполняем метод **schedule**, в котором запускается задача.

Метод **schedule** проверяет, что задача уже создана и отменяет ее. Далее планирует новую, с отложенным на 1000 мс запуском и периодом = **interval**. Т.е. можно сказать, что этот метод перезапускает задачу с использованием текущего интервала повтора (**interval**), а если задача еще не создана, то создает ее. Сама задача просто выводит в лог текст **run**. Если **interval** = 0, то никаких действий не предпринимается.

Метод **upInterval** получает на вход значение, увеличивает **interval** на это значение и перезапускает задачу. Соответственно задача после этого будет повторяться реже.

Метод **downInterval** получает на вход значение, уменьшает **interval** на это значение (но так, чтоб не меньше 0) и перезапускает задачу. Соответственно задача после этого будет повторяться чаще.

onBind возвращает **binder**. Это объект класса **MyBinder**.

MyBinder расширяет стандартный **Binder**, мы добавляем в него один метод **getService**. Этот метод возвращает нашу службу **MyService**.

Т.е. в подключаемом **Activity**, в методе **onServiceConnected** мы получим объект, который идет на выход метода **onBind**. Далее преобразуем его к типу **MyBinder** посредством вызова **getService**. В итоге в **Activity** будет ссылка на объект-службу **MyService**.

ОСНОВНАЯ ЛИТЕРАТУРА

1. Семакова, А. Введение в разработку приложений для смартфонов на ОС Android / А. Семакова. - 2-е изд., испр. - М. : Национальный Открытый Университет «ИНТУИТ», 2016. - 103 с. : ил. ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429181 (15.06.2017).

2. Введение в разработку приложений для ОС Android / Ю.В. Березовская, О.А. Юфрякова, В.Г. Вологдина и др. - 2-е изд., испр. - М. : Национальный Открытый Университет «ИНТУИТ», 2016. - 434 с. : ил. - Библиогр. в кн. ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=428937 (15.06.2017).

3. Разработка приложений для смартфонов на ОС Android / Е.А. Латухина, О.А. Юфрякова, Ю.В. Березовская, К.А. Носов. - 2-е изд., исправ. - М. : Национальный Открытый Университет «ИНТУИТ», 2016. - 252 с. : ил. - Библиогр. в кн. ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=428807 (15.06.2017).

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

1. Ретабоуил, С. Android NDK: руководство для начинающих. [Электронный ресурс] — Электрон. дан. — М. : ДМК Пресс, 2016. — 518 с. — Режим доступа: http://e.lanbook.com/book/82810 — Загл. с экрана.

2. Ёранссон, А. Эффективное использование потоков в операционной системе Android. [Электронный ресурс] — Электрон. дан. — М. : ДМК Пресс, 2015. — 304 с. — Режим доступа: http://e.lanbook.com/book/93268 — Загл. с экрана.

3. Соколова, В.В. Разработка мобильных приложений : учебное пособие / В.В. Соколова ; Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Томский государственный университет», Министерство образования и науки Российской Федерации. - Томск : Издательство Томского политехнического университета, 2015. - 176 с. : ил., табл., схем. - Библиогр. в кн.. - ISBN 978-5-4387-0369-3 ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=442808 (15.06.2017).

4. Баженова, И.Ю. Язык программирования Java / И.Ю. Баженова. - М. : Диалог-МИФИ, 2008. - 254 с. : табл., ил. - ISBN 5-86404-091-6 ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=54745 (15.06.2017).