УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«Брестский государственный университет имени А.С. Пушкина»

РАЗРАБОТКА ПРИЛОЖЕНИЙ ДЛЯ МОБИЛЬНЫХ ОПЕРАЦИОННЫХ СИСТЕМ "ANDROID"

Электронный курс лекций
Учебная программа учреждения высшего образования по учебной дисциплине для специальности высшего образования второй ступени (магистратуры):
1-31 81 06 «Веб-программирование и интернет-технологии» физико-математического факультета

Брест
БрГУ имени А.С. Пушкина
2015

Кафедра ПМиТП

Рецензенты:

заведующий кафедрой информатики и прикладной математики
учреждения образования «Брестский государственный технический университет»,
кандидат технических наук, доцент

С.И. Парфомук

Доцент кафедры алгебры, геометрии и математического моделирования
учреждения образования «Брестский государственный университет имени
А.С. Пушкина»,
кандидат технических наук, доцент

Е.Е. Пролиско

Кондратюк, А.П.

Разработка приложения для мобильных операционных систем "Android": электрон. курс лекций для студ. второй ступени (магистратуры) специальности 1-31 81 06 «Веб-программирование и интернет-технологии» физ.-мат. фак. / А.П. Кондратюк ; Брест. гос. ун-т им. А.С. Пушкина, каф. ПМ и ТП. – Брест : электрон. издание БрГУ, 2015. – 219 с.

Электронный курс лекций написан в соответствии с действующей базовой программой по дисциплине «Разработка приложений для мобильных операционных систем "Android"» и ставит своей целью облегчить самостоятельную работу студентов с теоретическим материалом при подготовке к лекциям, практическим занятиям и зачету.

Предназначено для студентов специальности 1-31 81 06 «Веб-программирование и интернет-технологии».

СОДЕРЖАНИЕ

Кафедра ПМиТП

ПРЕДИСЛОВИЕ

Настоящий электронный курс лекций предназначен для студентов специальности 1-31 81 06 «Веб-программирование и интернет-технологии» физико-математического факультета. Он написан в соответствии с действующей базовой программой по дисциплине «Разработка приложений для мобильных операционных систем "Android"».

В электронном издании излагается теоретический материал, содержащий вопросы: устройство платформы Android, обзор сред программирования и эмулятров, основные виды Android-приложений и их безопасность, архитектура приложения и его компоненты, манифест и ресурсы приложения, основы разработки многооконных мобильных приложений и их интерфейсов, использование возможностей смартфона, обзор библиотек Android, работа с базами данных и мультимедиа. Теоретический материал иллюстрируется примерами.

Курс лекций ставит своей целью облегчить самостоятельную работу студентов с теоретическим материалом при подготовке к лекциям, лабораторным занятиям и зачету.

Автор.

ГЛАВА 1
Программное обеспечение

§1. Введение

Аннотация: Целью лекции является описание основных принципов разработки для ОС Android. В лекции рассказывается об устройстве платформы Android, приводится обзор сред программирования, описываются возможности отладки на эмуляторах и реальных устройствах. Имеется большое количество разнообразных примеров и иллюстраций. В конце приведен список дополнительных источников. Лекция является обязательной для понимания следующих тем курса.

Скриншоты приложений взяты из магазина приложений Google Play из магазина приложений Google Play или сделаны самостоятельно с использованием смартфона Мегафон SP-A20i Mint на платформе Intel Medfield.

Презентацию к данной лекции можно скачать здесь.

Android - *операционная система* для мобильных устройств: смартфонов, планшетных компьютеров, КПК. В настоящее время именно *Android* является самой широко используемой операционной системой для мобильных устройств. Подтверждение этого факта можно найти в таблице, составленной *по* данным аналитической компании Gartner.

Таблица 1.1. Мировые продажи смартфонов конечным пользователям, распределение по ОС

Операционная система	**Продано (тыс.ед.) III кв. 2013**	**Доля рынка (%) III кв. 2013**	**Продано (тыс.ед.) III кв. 2012**	**Доля рынка (%) III кв. 2012**
Android	205022,7	81,9	124552,3	72,6
iOS	30330,0	12,1	24620,3	14,3
Microsoft	8912,3	3,6	3993,6	2,3
BlackBerry	4400,7	1,8	8946,8	5,2
Bada	633,3	0,3	4454,7	2,6
Symbian	457,5	0,2	4401,3	2,6
другие	475,2	0,2	683,7	0,4
Общее колво:	**250231,7**	**100,0**	**171652,7**	**100,0**

Источник: Gartner (ноябрь 2013)

Внимательное изучение таблицы позволяет увидеть подавляющую популярность смартфонов под управлением ОС *Android* в мире, доля таких устройств не первый год превышает половину от общего числа купленных смартфонов. Кроме всего прочего, эта популярность продолжает расти. Очевидно, что армия пользователей смартфонов под управлением *Android* будет искать дополнительные приложения для своих устройств, в связи с этим умение разрабатывать эти самые приложения может принести много пользы своему владельцу. Например, можно разрабатывать

для себя полезные, интересные, занимательные (нужное подчеркнуть) приложения, а можно, разведав обстановку и осмотревшись, сделать разработку мобильных приложений своей профессиональной деятельностью, основной или дополнительной.

Курс "Разработка приложений для смартфонов на ОС *Android*" предоставляет возможность приобрести начальные навыки разработки мобильных приложений, если остановиться только на первой его части. Изучение полной версии курса позволит сделать серьезный шаг к тому, чтобы профессионально разрабатывать мобильные приложения и получать от этой деятельности не только моральное, но и материальное удовлетворение.

Данная лекция является первой для всего курса, призвана ввести читателя в курс дела. В первую *очередь* в ней рассматриваются вопросы становления и развития ОС *Android*. Для успешного программирования под *Android* необходимо понимать внутреннюю организацию и архитектуру этой платформы, а также полезно знать, какие инструменты и среды разработки можно использовать. Этим вопросам посвящена основная часть лекции. Кроме того, в лекции рассматриваются особенности запуска и отладки мобильных приложений.

Немного истории

Рассмотрим, как все начиналось. В 2003 году в Пало Альто, штат Калифорния Энди Рубин с единомышленниками (Рич Майнер, Ник Сирс

и Крис Уайт) основали компанию Android Inc. Поначалу в компании занимались проектированием мобильных гаджетов, которые на основе геолокационных данных автоматически подстраивались под нужды пользователей.

В августе 2005 года Android Inc. стала дочерней компанией Google. Энди Рубин, Рич Майнер и Крис Уайт остались в Android Inc. и начали работать над операционной системой, базирующейся на ядре Linux. В Google задумали реализовать мощнейшую платформу, пригодную к использованию на тысячах различных моделей телефонов. В связи с этим был создан Open Handset Alliance (OHA) - консорциум, состоящий из более 80 компаний, направляющий свои усилия на разработку открытых стандартов для мобильных устройств. В состав OHA входят такие гиганты, как Google (организатор и идейный вдохновитель), HTC, Sony, Dell, Intel, Motorola, Qualcomm, Texas Instruments, Samsung Electronics, LG Electronics, T-Mobile, Sprint Corporation, NVIDIA и многие другие.

Первая версия Android была представлена 23 сентября 2008 года, версии было дано название Apple Pie (можно заметить созвучие с прямым конкурентом). Далее так повелось, что название каждой очередной версии представляет какой-либо десерт, при этом первые буквы наименований в порядке версий соответствуют буквам латинского алфавита по порядку. С развитием обновлений Android можно познакомиться в приведенной ниже таблице.

Версия, логотип, дата выхода	Основные возможности
Android 1.0 Apple Pie	Первый стабильный релиз, основан на ядре Linux 2.6.25. Поддерживается: файловая система FAT32, стек интернет-протоколов TCP/IP; протоколы передачи данных: 802.11 b/g Wi-Fi,Bluetooth 2.0 EDR, GPRS, EDGE, UMTS, HSDPA; фото и видео съемка, однако недостаточно опций для настройки разрешения камеры, баланса белого и др.; сенсорные дисплеи и landscape режим отображения данных на экране, максимальная цветность дисплея - 16 бит (тип HVGA); виджеты и ярлыки на рабочем столе (Home Screen), сменные обои; регулярные телефонные функции, контроль вызова, конференц-связь, легкая интеграция с контактами; полноценный web-браузер на движке WebKit, HTML, XHTML; e-mail клиент, протоколы POP3, IMAP4, SMTP;

	медиа проигрыватель, позволяющий управлять, импортировать, проигрывать медиа контент в различных форматах. Базовые приложения: будильник; калькулятор; календарь; камера; контакты; сообщения (в том числе MMS); настройки; голосовой набор. Минимальные системные требования для запуска и работы: архитектура ARM, 128 MB RAM, 256 MB ROM. Видео презентация: здесь
Android 1.1 Banana Bread февраль 2009 (API level: 2)	Нововведения: Исправлены проблемы: с будильником; со спящим режимом; с вызовом дисплея набора номера; в IMAP ошибки запроса пароля и др. Изменения API. Добавлены подробности и отзывы к картам. Добавлена поддержка вложений из MMS. Локализации: Английская US (en_ US) Немецкая (de) Подробности: здесь

Android 1.5 Cupcake апрель 2009 (API level: 3)	Нововведения: Поддержка экранной клавиатуры (); акселерометра; видеозапись и воспроизведение видео; приложение для работы с YouTube; стерео Bluetooth; функция копирования и вставки между приложениями (copy& paste). Локализации: добавились очень многие, в том числе и русская (ru_RU). Система: новое Linux ядро (версия 2.6.27); автоматическая проверка и восстановление файловой системы на SD card; новое приложение для просмотра CTK меню оператора (SIM Application Toolkit 1.0). Изменения в пользовательском интерфейсе (UI): изменено большинство UI-элементов, добавлены новые виджеты; определение режима (книжный или портретный) работы программы; анимированное переключение между окнами. Подробности: здесь

Android 1.6 Donut сентябрь 2009 (API level: 4)	Нововведения: Система: новое ядро Linux (версия 2.6.29); поддержка сотового стандарта CDMA; поддержка разрешений дисплеев: QVGA и WVGA; обновленный медиа-движок OpenCore 2; движок синтеза речи (многоязыковой); Gesture Builder поддержка возможности (для разработчиков) создавать, сохранять, загружать и распознавать жесты, прикреплять к определенным действиям. Пользовательские возможности: строка быстрого поиска (прямо с рабочего стола): история и закладки в браузере, контакты и поиск в интернете; возможность подключаться к видам VPN: L2TP/IPSEC pre-shared key based VPN, L2TP/IPSEC certificate based VPN, L2TP only VPN, PPTP only VPN; ускорение работы камеры; индикатор работы батареи позволяет увидеть сколько энергии потребляют работающие программы и сервисы. Обновленный Android Market. Подробности: здесь

Android 2.0, 2.0.1, 2.1 Eclair октябрь 2009 (API level: 5) (API level: 6) (API level: 7)	Нововведения в *2.0*: поддержка работы нескольких почтовых аккаунтов одновременно, возможность использования совместных папок (входящие, исходящие) для всех аккаунтов; быстрый способ работы с контактами Quick Contact; поиск по всем сохраненным SMS и MMS сообщениям, удаление старых после заданного срока; возможности камеры: вспышка, цифровой зум, сценические режимы, баланс белого, цветовые эффекты, макрофокусировка; улучшенное расположение виртуальных клавиш клавиатуры, поддержка комбинированных нажатий клавиш (технология мультитач), усовершенствованная функция автодополнения; поддержка HTML5, версии Bluetooth 2.1, новых профилей OPP и PBAP. Подробности:здесь Нововведения *2.0.1*: подрелиз версии 2.0, включающий в себя незначительные изменения в функционале и по большей части bugfix-ом версии 2.0. Подробности: здесь

	Нововведения *2.1*: основным новшеством, представляющим интерес для конечного пользователя, стало добавление анимированных (живых) обоев, остальные изменения в Framework API, представляют интерес для разработчиков. Подробности: здесь
Android 2.2 Froyo май 2010 (API level: 8)	Нововведения: рост производительности примерно в 3-5 раз за счет использования Dalvik Virtual Machine Just-in-Time компилятора; возможности установки приложений на SD-карту, переноса приложений из внутренней памяти на карту и обратно; возможность использовать смартфон в качестве точки доступа к интернету, в качестве модема для других устройств; поддержка Adobe Flash; V8 javascript существенно повысил скорость работы штатного браузера. Подробности: здесь

Android 2.3, 2.3.3 Gingerbread декабрь 2010 (API level: 9) (API level: 10)	До весны 2013 года самая массовая версия на рынке. Нововведения: новое ядро Linux 2.6.35; поддержка открытых мультимедийных стандартов (VP8 и WebM), форматов ACC/AMR, звуковых эффектов и эквалайзера, фронтальной камеры (интеграция с VOIP(SIP)); обновленный GUI: уменьшение времени доступа к функциям, повышение общей энергоэффективности системы; улучшение стандартной клавиатуры системы: поддержка словарей, технологии мультитач, упрощенное выделение и копирование текста; поддержка технологии NFC; расширение возможностей работы с датчиками положения телефона. Подробности:здесь

Android 3.0-3.2 Honeycomb февраль 2011 (API level: 11) (API level: 12) (API level: 13)	Специальная версия для работы на планшетах (MID, tablets). Нововведения 3.0: новое ядро Linux 2.6.36; поддержка файловой системы ext4, файловой системы FUSE для MTP устройств; поддержка режима USB-хост для работы с клавиатурой, мышью и USB-хабами; поддержка MTP/PTP; виртуальная машина Dalvik: поддержка и оптимизация SMP, множество улучшений JIT, улучшенный сборщик мусора; совершенно новый интерфейс с полноценной оптимизацией под устройства с большими экранами; поддержка виртуальных рабочих столов, каждый из которых может иметь свой набор виджетов и ярлыков; улучшенные и переработанные базовые приложения: Browser, e-mail и др. Подробности: здесь Нововведения 3.1: поддержка работы кардридера; усовершенствован GUI: доработан менеджер задач, позволяющий переключаться между множеством различных приложений (в 3.0 только 5 программ

	одновременно); возможность менять размер виджетов, как по горизонтали, так и по вертикали. Подробности: здесь Нововведения 3.2: расширен спектр поддерживаемых планшетов; возможность автоматического масштабирования приложений для отображения на более крупных экранах. Подробности: здесь
Android 4.0, 4.0.3 Ice Cream Sandwich ноябрь 2011 (API level: 14) (API level: 15)	Нововведения: поддержка и смартфонов, и планшетов; поддержка новых процессорных архитектур, помимо ARM поддержка Intel x86 и MIPS; возможность разблокировки экрана: при помощи функции определения лица; жестами: перетащить замочек из центра экрана на иконку приложения и оно запустится; многозадачность: кнопка Recent Apps позволяет мгновенно переходить от одной задачи к другой с помощью списка в системной панели; новые элементы управления передачей данных через сеть: в приложении Настройки можно увидеть общее использование данных по каждому типу сети,

	объем данных, используемых каждым работающим приложением; доступность Android 4.0 для слепых и слабовидящих пользователей, браузер поддерживает экранного чтеца, который воспроизводит все видимое активное содержимое на экране; AndroidBeam - удобное средство обмена между двумя NFC-устройствами; Wi-Fi Direct и Bluetooth H DP, HFP: возможность прямого подключения к соответствующим устройствам. Подробности: здесь
Android 4.1-4.3 Jelly Bean июль 2012 (API level: 16) (API level: 17) (API level: 18)	Нововведения 4.1: увеличена скорость прорисовки интерфейса, улучшен поиск, добавлено несколько полезных сервисов; улучшена работа со словарями, возможно использовать голосовой ввод без подключения к интернету; специальные возможности: возможность управления смартфоном с помощью жестов и голосовых подсказок, подключения устройств ввода, поддерживающих шрифт Брайля; существенно доработана функция передачи данных Beam; переработан поиск (вместо ссылок ответ на

запрос); голосовой поиск; Google Now: нужная информация в нужное время.
Подробности: здесь
Нововведения 4.2:
реализована поддержка нескольких пользователей (планшеты); поддержка wireless display: возможность трансляции видео и изображений на внешний экран;
возможность отображения полезной информации в режиме сна, при подключении к док-станции или на зарядке; улучшена панель уведомлений.
Подробности: здесь
Нововведения 4.3:
ускорение работы системы; более точный набор на клавиатуре; скрытая возможность управления процессами программ (необходима активация); поддержка OpenGL/ES 3.0 (не на всех устройствах).
Подробности: здесь

Android 4.4 Kit Kat октябрь 2013 (API level: 19)	Нововведения: многозадачность, оптимизация распределения ресурсов между приложениями; определитель номера работает не только с адресной книгой (например, Google maps); серьезная интеграция приложения Hangouts (отправка SMS, MMS, голосовые и видеовызовы); в состав вошел Quickoffice, интегрированный с Google Drive; поддержка принтеров, подключение через приложения поддерживающие печать (например, Google Cloud Print, HP ePrint); поддержка стандарта Wi-Fi Miracast, позволяющий вещать изображение на телевизор; возможность захвата экрана для записи видео. Подробности: здесь

Таблица 1.2. История обновлений ОС Android

§2. Устройство платформы Android

Платформа *Android* объединяет операционную систему, построенную на основе ядра ОС Linux, промежуточное *программное обеспечение* и встроенные мобильные приложения. Разработка и развитие мобильной платформы *Android* выполняется в рамках проекта AOSP *(Android Open Source Project)* под управлением OHA *(Open Handset Alliance)*, руководит всем процессом поисковый гигант Google.

Android поддерживает фоновое выполнение задач; предоставляет богатую библиотеку элементов пользовательского интерфейса; поддерживает 2D и 3D графику, используя OpenGL стандарт; поддерживает *доступ* к файловой системе и встроенной базе данных SQLite.

С точки зрения архитектуры, система *Android* представляет собой полный программный *стек*, в котором можно выделить следующие уровни:

- **Базовый уровень (Linux Kernel)** - уровень абстракции между аппаратным уровнем и программным стеком;
- **Набор библиотек и среда исполнения (Libraries & Android Runtime)** обеспечивает важнейший базовый функционал для приложений, содержит виртуальную машину Dalvik и базовые библиотеки Java необходимые для запуска Android приложений;
- **Уровень каркаса приложений (Application Framework)** обес-

печивает разработчикам доступ к API, предоставляемым компонентами системы уровня библиотек;

- **Уровень приложений (Applications)** - набор предустановленных базовых приложений.

Наглядное изображение архитектуры на рисунке 1.

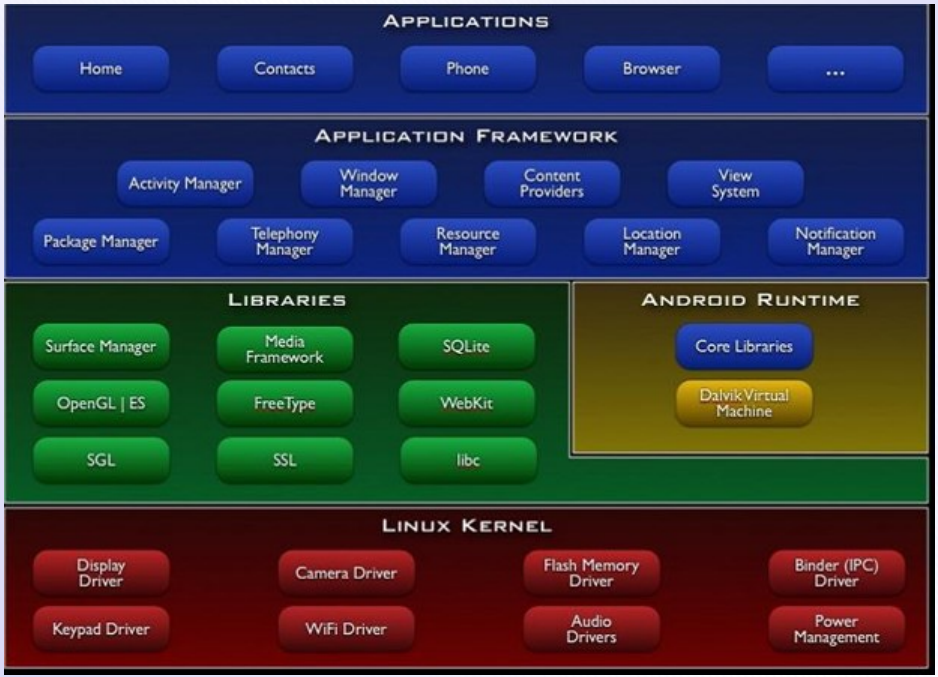


Рисунок 1. Архитектура Android

Рассмотрим компоненты платформы более подробно.

В основании компонентной иерархии лежит *ядро* ОС Linux 2.6 (несколько урезанное), служит промежуточным уровнем между аппаратным и программным обеспечением, обеспечивает функционирование системы, предоставляет системные службы ядра: *управление памятью*, энергосистемой и процессами, обеспечение безопасности, работа с сетью и драйверами.

Уровнем выше располагается набор библиотек и среда исполнения. Библиотеки реализуют следующие функции:

- предоставляют реализованные алгоритмы для вышележащих уровней;
- обеспечивает поддержку файловых форматов;
- осуществляет кодирование и декодирование информации (например, мультимедийные кодеки);
- выполняет отрисовку графики и т.д.

Библиотеки реализованы на C/C++ и скомпилированы под конкретное *аппаратное обеспечение* устройства, вместе с которым они и поставляются производителем в предустановленном виде.

Рассмотрим некоторые библиотеки:

Surface Manager - композитный менеджер окон. Поступающие команды отрисовки собираются в закадровый буфер, где они накапливаются, составляя некую композицию, а потом выводятся на экран. Это позволяет системе создавать интересные бесшовные эффекты, прозрачность окон и плавные переходы.

Media Framework - библиотеки, реализованные на базе PacketVideo OpenCORE. Используются для записи и воспроизведения аудио и видео контента, а также для вывода статических изображений. Поддерживаются форматы: MPEG4, H.264, MP3, AAC, AMR, JPG и PNG.

SQLite - легковесная и производительная реляционная СУБД, используется в Android в качестве основного движка для работы с базами данных.

3D библиотеки - используются для высокооптимизированной отрисовки 3D-графики, при возможности используют аппаратное ускорение. Библиотеки реализованы на основе API OpenGL|ES. OpenGL|ES (OpenGL for Embedded Systems) - подмножество графического программного интерфейса OpenGL, адаптированное для работы на встраиваемых системах.

FreeType - библиотека для работы с битовыми картами, для растеризации шрифтов и осуществления операций над ними.

LibWebCore - библиотеки браузерного движка WebKit, используемого также в известных браузерах Google Chrome и Apple Safari.

SGL (Skia Graphics Engine) - открытый движок для работы с 2D-графикой. Графическая библиотека является продуктом Google и часто используется в других программах.

SSL - библиотеки для поддержки одноименного криптографического протокола.

Libc - стандартная библиотека языка C, а именно ее BSD реализация, настроенная для работы на устройствах на базе Linux.

Среда исполнения включает в себя библиотеки ядра, обеспечивающие большую часть низкоуровневой функциональности, доступной библиотекам ядра языка *Java*, и виртуальную машину Dalvik, позволяющую запускать приложения. Каждое *приложение* запускается в своем экземпляре виртуальной машины, тем самым обеспечивается изоляция работающих приложений от ОС и друг от друга. Для исполнения на виртуальной машине Dalvik *Java*-классы компилируются в исполняемые файлы с расширением .dex с помощью инструмента dx, входящего

в состав *Android* SDK. DEX (Dalvik EXecutable) - формат исполняемых файлов для виртуальной машины Dalvik, оптимизированный для использования минимального объема памяти. При использовании *IDE* Eclipse и плагина *ADT(Android Development Tools) компиляция* классов *Java* в формат .dex происходит автоматически.

Архитектура Android Runtime такова, что работа программ осуществляется строго в рамках окружения виртуальной машины, что позволяет защитить *ядро* ОС от возможного вреда со стороны других ее составляющих. Поэтому код с ошибками или *вредоносное ПО* не смогут испортить *Android* и устройство на его базе, когда сработают.

На еще более высоком уровне располагается каркас приложений *(Application Framework), архитектура* которого позволяет любому приложению использовать уже реализованные возможности других приложений, к которым разрешен *доступ*. В состав каркаса входят следующие компоненты:

- богатый и расширяемый набор представлений **(Views)**, который может быть использован для создания визуальных компонентов приложений, например, списков, текстовых полей, таблиц, кнопок или даже встроенного web-браузера;
- контент-провайдеры **(Content Providers)**, управляющие данными, которые одни приложения открывают для других, чтобы те могли их использовать для своей работы;

- менеджер ресурсов **(Resource Manager)**, обеспечивающий доступ к ресурсам без функциональности (не несущим кода), например, к строковым данным, графике, файлам и другим;
- менеджер оповещений **(Notification Manager)**, позволяющий приложениям отображать собственные уведомления для пользователя в строке состояния;
- менеджер действий **(Activity Manager)**, управляющий жизненными циклами приложений, сохраняющий историю работы с действиями, предоставляющий систему навигации по действиям;
- менеджер местоположения **(Location Manager)**, позволяющий приложениям периодически получать обновленные данные о текущем географическом положении устройства.

Application Framework предоставляет в распоряжение приложений в ОС *Android* вспомогательный функционал, благодаря чему реализуется принцип многократного использования компонентов приложений и ОС. Естественно, в рамках политики безопасности.

И, наконец, самый высокий, самый близкий к пользователю уровень приложений. Именно на этом уровне *пользователь* взаимодействует со своим устройством, управляемым ОС *Android*. Здесь представлен набор базовых приложений, который предустановлен на ОС *Android*. Например, *браузер, почтовый клиент, программа* для отправки *SMS*, карты,

календарь, *менеджер* контактов и др. *Список* интегрированных приложений может меняться в зависимости от модели устройства и версии *Android*. К этому уровню также относятся все пользовательские приложения.

Разработчик обычно взаимодействует с двумя верхними уровнями архитектуры *Android* для создания новых приложений. Библиотеки, система исполнения и *ядро* Linux скрыты за каркасом приложений.

Повторное использование компонентов других приложений приводит к идее задач в *Android*. Приложение может использовать компоненты другого *Android* приложения для решения задачи, например, если разрабатываемое *приложение* предполагает использование фотографий, оно может вызвать *приложение*, управляющее фотографиями и зарегистрированное в системе *Android*, выбрать с его помощью фотографию и работать с ней.

Для пополнения коллекции приложений своего мобильного устройства *пользователь* может воспользоваться приложением Google Play, которое позволяет покупать и устанавливать приложения с сервиса Google Play. Разработчики, в свою *очередь*, могут выкладывать свои приложения в этот сервис, Google Play отслеживает появление обновлений приложения, сообщает пользователям этого приложения об обновлении и предлагает установить его. Также Google Play предоставляет разработчикам *доступ* к услугам и библиотекам, например, *доступ* к использованию и отображению Google Maps.

Для установки приложения на устройствах с ОС *Android* создается *файл* с расширением *.apk *(Android package)*, который содержит исполняемые файлы, а также вспомогательные компоненты, например, файлы с данными и *файлы ресурсов*. После установки на устройство каждое *приложение* "живет"в своем собственном изолированном экземпляре виртуальной машины Dalvik.

§3. Обзор сред программирования

Прежде чем начать разрабатывать приложения под *Android*, рассмотрим существующие инструменты, подходящие для этих целей. Можно выделить необходимые инструменты, без которых разработка мобильных приложений под *Android* просто невозможна. С другой стороны, существует большое количество вспомогательных систем, в какой-то мере упрощающих процесс разработки.

К обязательным инструментам относится *Android SDK* - набор средств программирования, который содержит инструменты, необходимые для создания, компиляции и сборки мобильного приложения. Рассмотрим кратко наиболее важные инструменты, входящие в состав *Android SDK*:

- **SDK Manager** - инструмент, позволяющий загрузить компоненты Android SDK. Показывает пакеты Android SDK и их статус: установлен (Installed), не установлен (Not Installed), доступны обновления (Update available).

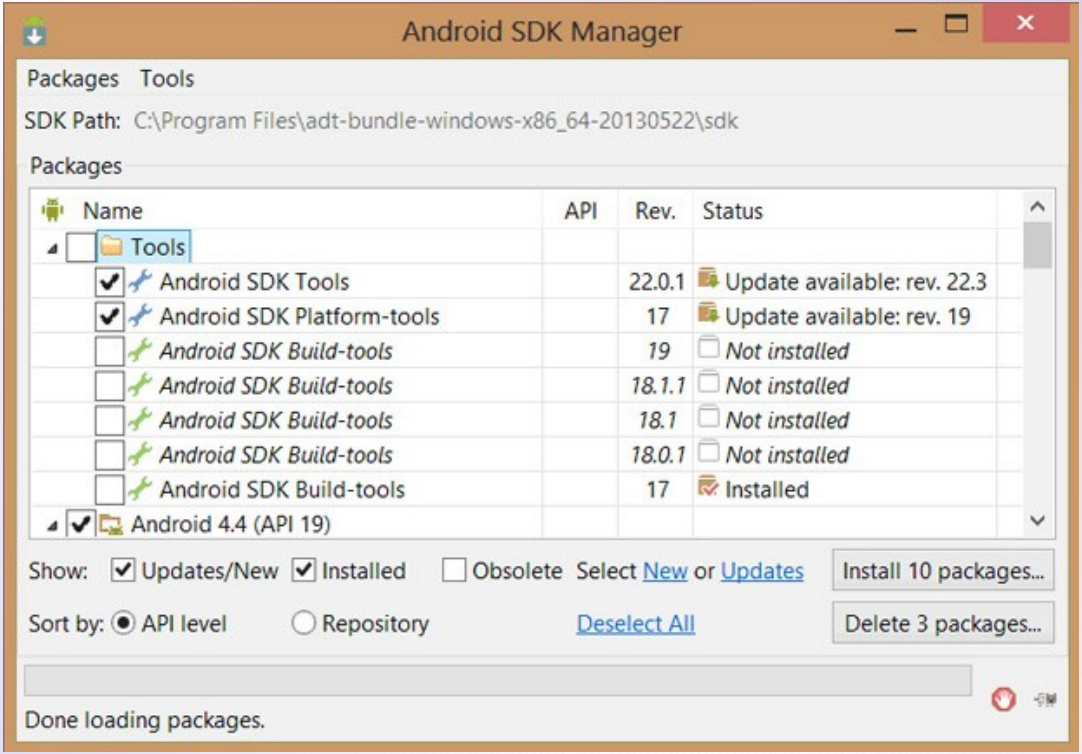


Рисунок 1. Android SDK Manager

- **Debug Monitor** - самостоятельный инструмент, предоставляющий графический интерфейс к нескольким инструментам, предназначенным для анализа и отладки Android приложений:

  – DDMS (Dalvik Debug Monitor Server) предоставляет услуги пе-

реброса портов, захват экрана устройства, информацию о потоках и динамической памяти устройства, вывод информации о действиях Android в реальном времени (logcat) и многое другое.

– Hierarchy Viewer позволяет отлаживать и оптимизировать пользовательский интерфейс Android приложения.

– Tracer for OpenGL ES - инструмент для анализа OpenGL|ES кода, используемого в мобильном приложении, позволяет захватывать команды OpenGL|ES и демонстрировать их по отдельным кадрам, что помогает понять как исполняются графические команды.

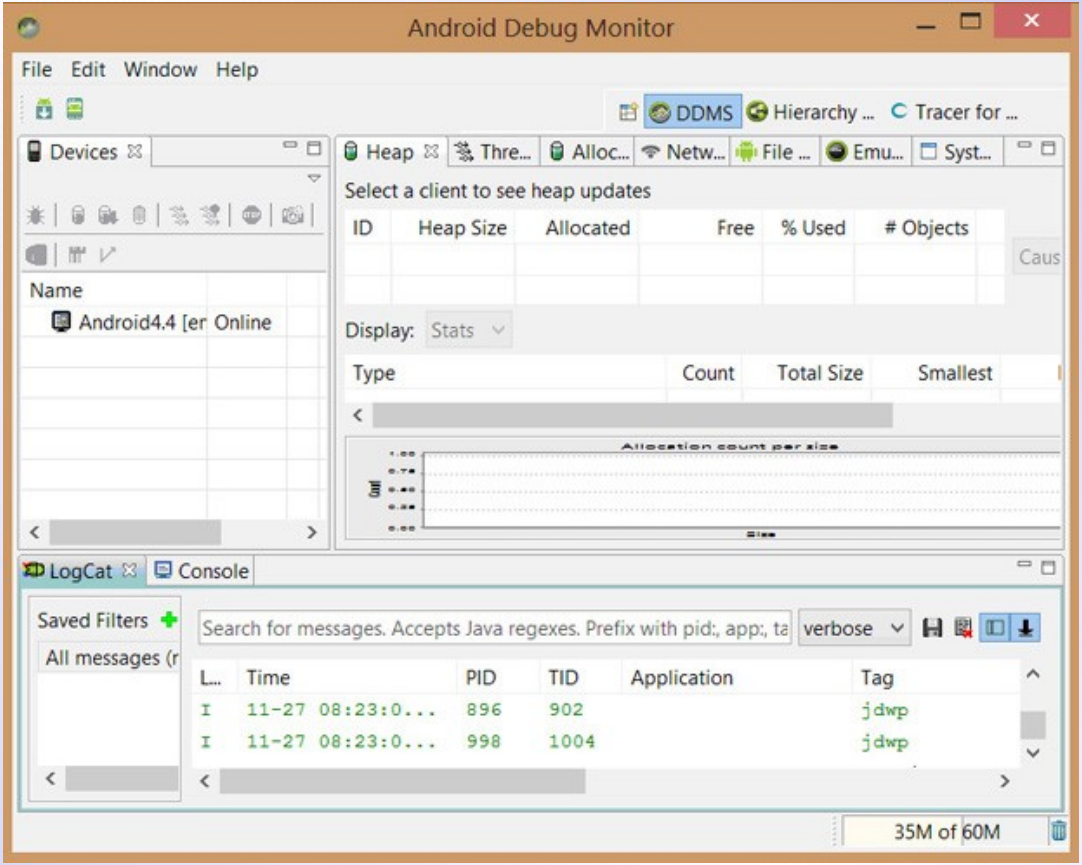


Рисунок 2. Окно инструмента Monitor

- **Android Emulator (emulator)** - виртуальное мобильное устройство, которое создается и работает на компьютере разработчика,

используется для разработки и тестирования мобильных приложений без привлечения реальных устройств.

- **AVD Manager** - предоставляет графический интерфейс для создания виртуальных Android устройств (AVDs), предусмотренных Android Emulator, и управления ими.

  (В ЛР№1 подробно рассматривается создание и использование виртуального устройства).

- **Android Debug Bridge (adb)** - гибкий инструмент, позволяющий управлять состоянием эмулятора или реального Android устройства, подключенного к компьютеру. Также может использоваться для установки Android приложения (.apk файл) на реальное устройство.

Мы рассмотрели основные инструменты, входящие в состав *Android SDK*, разумеется, не все и недостаточно подробно. Для более серьезного изучения инструментов имеет смысл обратиться к сайту разработчиков здесь. Для разработки мобильных приложений под *Android* уверенного владения инструментами из *SDK* вполне достаточно. Если же возникают какие-то вопросы, дополнительные инструкции *по* созданию проектов, компиляции, запуску из командной строки содержатся в руководстве от Google здесь.

В современных условиях разработка *ПО* в большинстве случаев ведется с использованием интегрированных сред разработки *(IDE)*. *IDE*

имеют несомненные достоинства: процесс компиляции, сборки и запуска приложения обычно автоматизирован, в связи с чем для начинающего разработчика создать свое первое *приложение* труда не составляет. Но чтобы заниматься разработкой всерьез, необходимо потратить силы и время на изучение возможностей самой среды. Рассмотрим IDE, пригодные для разработки под Android1. Для начала поговорим о двух средах разработки, которые рекомендует Google: *Android IDE (ADT)* и *Android Studio*.

Android IDE - *среда разработки* под *Android*, основанная на Eclipse. Предоставляет интегрированные инструменты для разработки, сборки и отладки мобильных приложений. В данном курсе *Android IDE* выбрана в качестве основной среды разработки. Возможности этой среды более подробно рассмотрены в первой лабораторной работе. Также там даны рекомендации поустановке и настройке среды, созданию и запуску первого приложения как на эмуляторе, так и на реальном устройстве.

Android Studio - *среда разработки* под *Android*, основанная на IntelliJ *IDEA*. Подобно *Android IDE*, она предоставляет интегрированные инструменты для разработки и отладки. Дополнительно ко всем возможностям, ожидаемым от IntelliJ, в *Android Studio* реализованы:

- поддержка сборки приложения, основанной на Gradle;
- специфичный для Android рефакторинг и быстрое исправление дефектов;

- lint инструменты для поиска проблем с производительностью, с юзабилити, с совместимостью версий и других;
- возможности ProGuard (утилита для сокращения, оптимизации и обфускации кода) и подписи приложений;
- основанные на шаблонах мастера для создания общих Android конструкций и компонентов;
- WYSIWYG редактор, работающий на многих размерах экранов и разрешений, окно предварительного просмотра, показывающее запущенное приложение сразу на нескольких устройствах и в реальном времени;
- встроенная поддержка облачной платформы Google.

Загрузить последнюю версию *Android Studio*, а также получить рекомендации по установке, настройке и началу работы можно здесь.

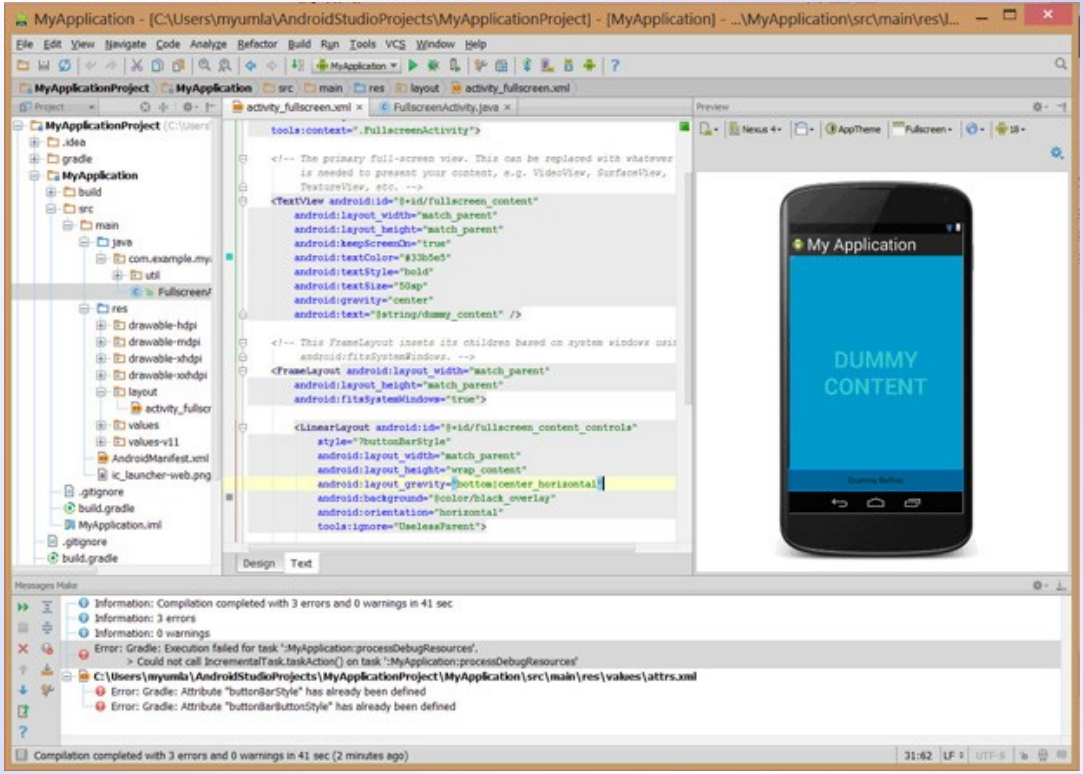


Рисунок 3. Среда разработки Android Studio

Перейдем к рассмотрению других инструментов, пригодных для разработки мобильных приложений под *Android*. Начнем с инструментов от Intel - Intel XDK и Intel Beacon Mountain.

Intel XDK позволяет легко разрабатывать кроссплатформенные мо-

бильные приложения; включает в себя инструменты для создания, отладки и сборки *ПО*, а также эмулятор устройств; поддерживает разработку для *Android*, Apple iOS, Microsoft *Windows8*, Tizen; поддерживает языки разработки: HTML5 и JavaScript.

Последняя тема данного курса полностью посвящена изучению нового поколения инструментальных средств разработки мобильных HTML5-приложений и Intel XDK, предполагается разработка мобильного приложения с использованием этих инструментов.

Intel Beacon Mountain - *среда разработки*, позволяющая создавать приложения для устройств, работающих под управлением ОС *Android*. Предоставляет инструменты необходимые для проектирования, разработки, отладки и оптимизации приложений под *Android*. Освобождает разработчика от необходимости поддерживать систему разработки в актуальном состоянии, следит за обновлениями и добавляет их в среду разработки по мере появления. Поддерживает разработку для целевых платформ на основе процессоров Intel *Atom* и *ARM*. Beacon Mountain построена на основе *Android IDE* (Eclipse, *Android ADT, Android SDK*), для более серьезной разработки и оптимизации добавлены следующие инструменты Intel:

- **Intel* Hardware Accelerated Execution Manager (Intel* HAXM)** - аппаратно поддерживаемый процессор виртуализации, использующий технологию виртуализации Intel* (Intel* VT) для ускорения

работы эмулятора в среде разработки.

- **Intel* Graphics Performance Analyzers (Intel* GPA) System Analyzer** поддерживает мобильные устройства с процессором Intel Atom под управлением ОС Android. Позволяет разработчикам оптимизировать загруженность системы при использовании процедур OpenGL, предоставляя возможность получать множество системных метрик в реальном времени, отображающих загруженность CPU, GPU и OpenGL ES API. Разработчик может запустить несколько графических экспериментов для выявления узких мест в обработке графики.
- **Intel* Integrated Performance Primitives (Intel* IPP) Preview** - библиотека оптимизированной обработки данных и изображений, поддерживающая мобильные устройства с платформой Intel под управлением ОС Android. Preview версия является частью полной версии Intel IPP, которая тоже поддерживает ОС Android.
- **Intel* Threading Building Blocks (Intel* TBB)** - широко используемая, признанная библиотека шаблонов C++ для создания масштабируемых приложений и увеличения производительности. Поддерживает мобильные устройства с платформой Intel под управлением Android. Проверенные алгоритмы позволяют разработчикам эффективно распараллелить C++ мобильные приложения, что повышает производительность при снижении энергетических за-

трат.

Загрузить *Intel Beacon Mountain* можно *по* ссылке здесь.

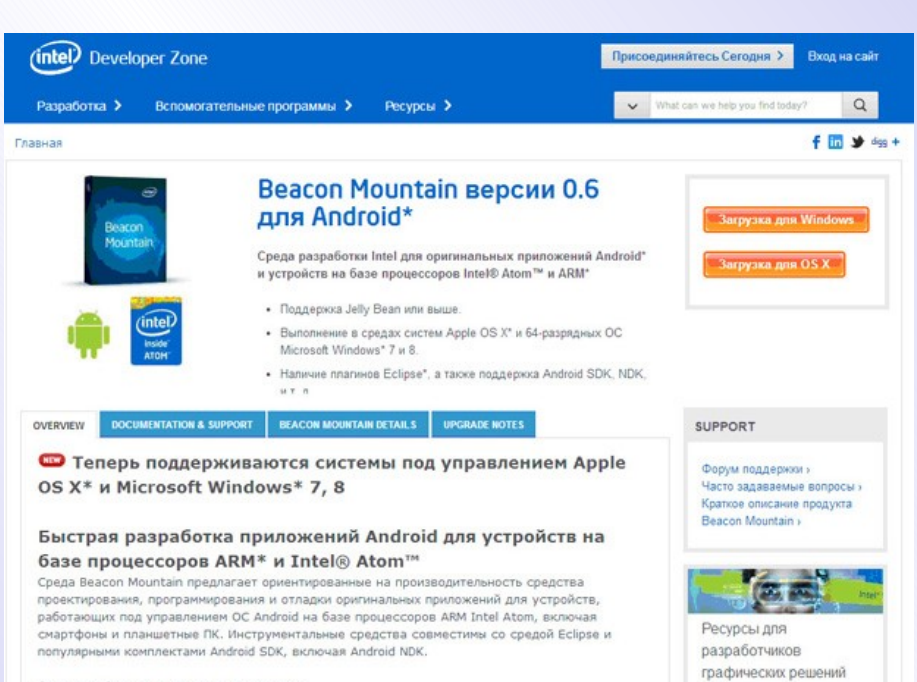


Рисунок 4. Страница поддержки Intel* Beacon Mountain

Нельзя обойти вниманием *инструментарий* Marmalade SDK.

Marmalade SDK - кроссплатформенное *SDK* от Ideaworks3D Limited. Представляет собой набор библиотек, образцов, инструментов и документации, необходимых для разработки, тестирования и развертывания приложений для мобильных устройств. Используется, в основном, для разработки игр. Многие получившие признание игры, такие как *Cut* the Rope и Plants vs. Zombies, были разработаны с использованием этого программного средства. К сожалению, Marmalade *SDK* представляет собой проприетарное *программное обеспечение* (самая дешевая *лицензия* $15 в месяц) и не может быть рекомендована в данном учебном курсе, но читатель может самостоятельно попробовать бесплатную 30-дневную версию, доступную *по* ссылке здесь.

Нельзя не сказать об отечественных разработках. Например, компания 1С идет в ногу со временем, версия платформы 1С 8.3 позволяет разрабатывать мобильные приложения. *Программный продукт* **"1С: Предприятие 8. Расширение для карманных компьютеров"** обеспечивает возможность работы с данными информационных баз 1С: Предприятия 8 на мобильных устройствах (карманных компьютерах, коммуникаторах, терминалах сбора данных), а также на персональных компьютерах (в том числе ноутбуках), не имеющих прямого доступа к информационным базам 1С:Предприятия 8.

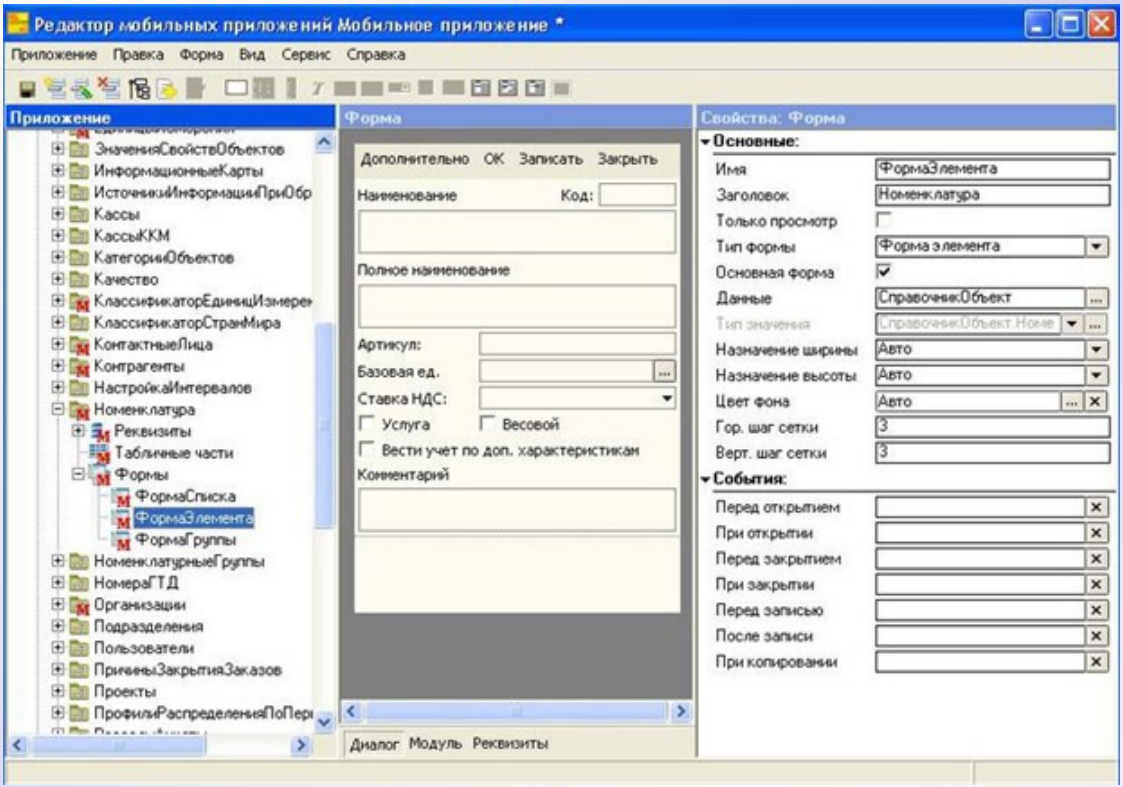


Рисунок 5. Редактор мобильных приложений 1С

Разумеется, данный *программный продукт* имеет очень узкую сферу применения, однако в некоторых случаях может являться наиболее удачным решением. Подробности *по* ссылке здесь.

Эмуляция. Стандартный эмулятор Android

Эмуляция (англ. *emulation*) в вычислительной технике - комплекс программных, аппаратных средств или их сочетание, предназначенное для копирования (или *эмулирования*) функций одной вычислительной системы (*гостя*) на другой, отличной от первой, вычислительной системе (*хосте*) таким образом, чтобы эмулированное поведение как можно ближе соответствовало поведению оригинальной системы (*гостя*). Целью является максимально точное воспроизведение поведения в отличие от разных форм компьютерного моделирования, в которых имитируется поведение некоторой абстрактной модели Википедия.

Эмулятор - виртуальное мобильное устройство, которое запускается на компьютере. При помощи эмулятора можно разрабатывать и тестировать приложения без использования реальных устройств. На рисунке 1 приведен пример запущенного стандартного эмулятора. Подробно работа с эмуляторами рассмотрена в лабораторной работе.

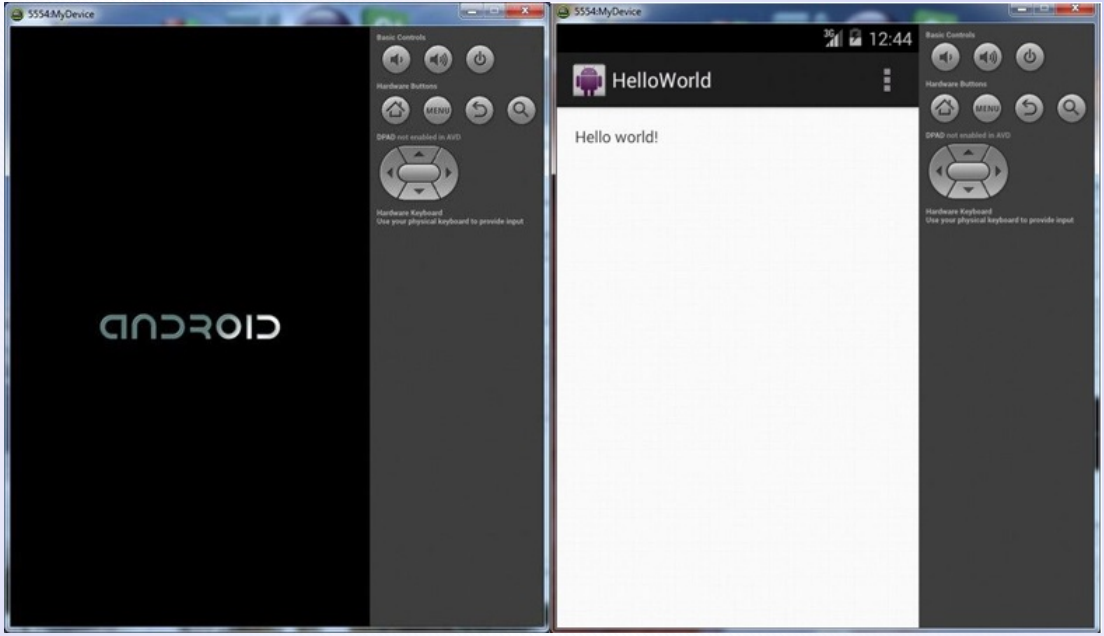


Рисунок 1. Эмулятор Android SDK в процессе запуска и приложение Hello, world!

К достоинствам использования *эмуляторов* можно отнести простоту их использования и нулевую стоимость. Разработчику не нужно покупать огромное количество устройств с различными характеристиками, чтобы проверить работоспособность приложения на различных смартфонах. Достаточно создать несколько эмуляторов с требуемыми характеристиками и запустить на них приложение. К сожалению, эмуляторы имеют и ряд недостатков:

- Требуют много системных ресурсов.
- Из-за различий в архитектуре процессоров компьютера и смартфона медленно запускаются. Современные персональные компьютеры построены на архитектурах x86 и x64, а большинство процессоров смартфонов на Android - ARM. Процесс эмуляции одной архитектуры на другой чрезвычайно сложен и происходит довольно медленно.
- В некоторых случаях стандартного *эмулятора* недостаточно. Речь идет о возможностях смартфонов, которыми обычные компьютеры не обладают (например, наличие датчика gps или акселерометра). В таких случаях полноценную отладку можно провести только с использованием реального устройства.

Альтернативные эмуляторы

Стандартный эмулятор, поставляемый вместе с Android SDK, не устраивает многих. Существуют проекты, поддерживающие разработку и развитие альтернативных эмуляторов. В качестве примера можно привести Genymotion (см. рис. 2) - быстрый эмулятор Android (по мнению его разработчиков). Он содержит предварительно настроенные образы Android (x86 с аппаратным ускорением OpenGL). Genymotion доступен для Linux, Windows и Mac OS X и требует для своей работы VirtualBox. Иными словами, Genymotion представляет собой виртуальную машину с установленной ОС Android, которую пользователь запускает так

же, как и другие виртуальные машины. Проблема высокого потребления системных ресурсов, конечно, не исчезает, однако скорость запуска существенно увеличивается.

В настоящее время активно развивается.

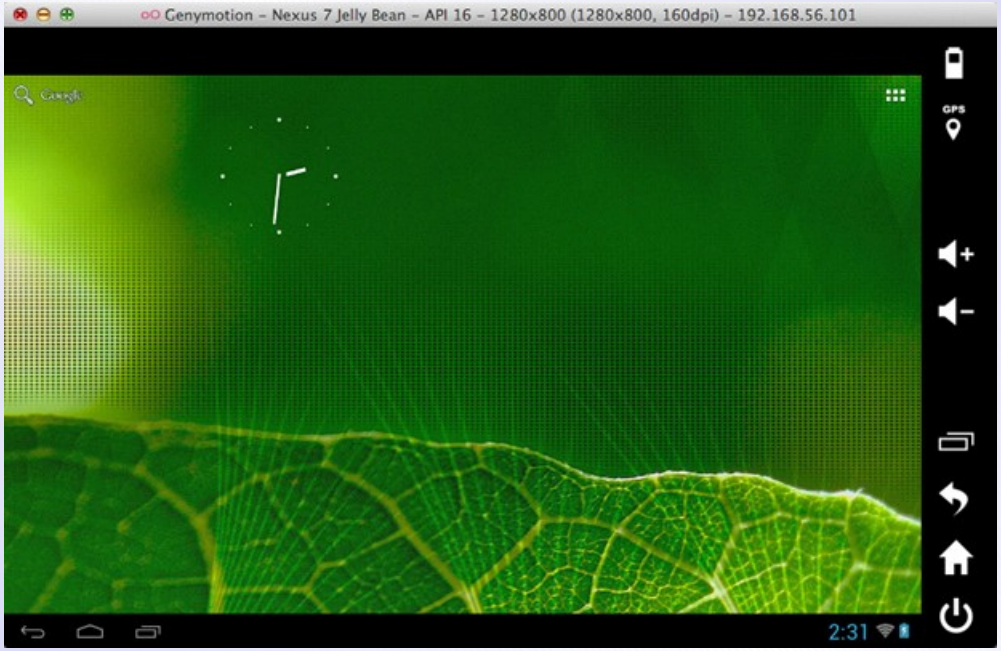


Рисунок 2. Альтернативный эмулятор Genymotion

ГЛАВА 2
Разработка приложений

§5. Возможности отладки на реальных устройствах

Разработанное *приложение* можно запустить на реальном устройстве, например, на смартфоне. Для этого необходимо проделать предварительную работу. Для запуска приложений, разработанных в *Android IDE*, необходимо:

- Настроить устройство (включить режим отладки по USB).
- Настроить компьютер (для Windows необходимо установить нужный драйвер вручную, нужны права администратора).
- Настроить среду и запустить проект на устройстве.

Подробности отладки на реальных устройствах описаны в лабораторной работе.

§6. Примеры приложений

Google Play - это магазин приложений от Google, позволяющий владельцам устройств с операционной системой *Android* устанавливать и приобретать различные приложения. Учётная *запись* разработчика, которая даёт возможность публиковать приложения, стоит $25. В настоящее время Google Play насчитывает более миллиона различных приложений, каждый месяц пользователями загружается несколько миллиардов. Разумеется, далеко не все из них высокого качества и поддерживаются разработчиками, встречается и вредоносное *программное обеспечение*.

В настоящий момент доступно более 30 различных категорий приложений. Внутри каждой категории приложения упорядочены на основании рейтинга, отзывов, количества скачиваний, страны распространения и других факторов.

Время от времени редакция Google Play собирает коллекции приложений или игр, основанных на теме или сезонном событии. Коллекции пользуются популярностью у клиентов за счет своевременности и актуальности.

Приведем примеры интересных и удобных приложений, заслуженно удерживающих высокие места в своих категориях уже долгое время.

Популярное игровое *приложение Cut* the Rope позволяет разобраться в правилах игры прямо в ее процессе и не требует чтения сложных

инструкций (см. рисунок 1). Идея игры предельно проста - в коробке сидит маленький зелёный монстр Ам Ням, которого надо кормить леденцами. Леденцы болтаются на веревках, и их надо правильно перерезать, чтобы леденец попал точно в рот Ам Няма. По ходу игры сложность уровней возрастает, появляются дополнительные препятствия. Попутно надо собирать звездочки, которые позволяют открывать новые локации.

Рисунок 1. Первый уровень игры Cut the Rope

Если вам нужен интерактивный помощник, который способен понимать сделанные устно указания и напоминать о делах в нужное время, *приложение* "Помнить все"(см. рис. 2) - то, что вам нужно. Используя библиотеку для распознавания речи, оно анализирует полученную информацию, отображает ее в виде текста, который при необходимости можно исправить, и устанавливает время для напоминания.

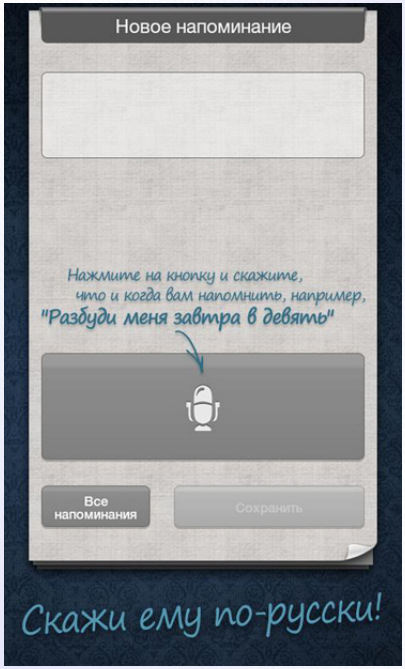


Рисунок 2. Приложение "Помнить все"распознает русскую речь

§7. Введение в разработку

Аннотация: В данной теме обсуждаются вопросы, связанные непосредственно с разработкой мобильных приложений для устройств, работающих под управлением Android. Рассматривается еще несколько общих вопросов: во-первых, какие виды мобильных приложений существуют и каковы особенности каждого вида; во-вторых, как организовано исполнение приложений в ОС Android и каким образом обеспечивается безопасная среда их функционирования. Понимание этих вопросов позволяет вести более осознанную разработку приложений. В лекции рассматривается архитектура Android приложений, основанная на идее многократного использования компонентов, которые являются основными строительными блоками. Подробно описываются основные компоненты, а также такие важные понятия для мобильных приложений, работающих под управлением Android, как манифест приложения и ресурсы.

В "Введение в разработку мобильних приложений"были рассмотрены общие вопросы, связанные с операционной системой *Android*, а также инструменты, используемые для разработки мобильных приложений. Изучены основы разработки интерфейсов мобильных приложений. В данной теме обсуждаются вопросы, связанные, непосредственно, с разработкой мобильных приложений для устройств, работающих под управлением *Android*.

Для начала предполагается рассмотреть еще несколько общих вопро-

сов: во-первых, какие виды мобильных приложений существуют и каковы особенности каждого вида; во-вторых, как организовано *исполнение* приложений в ОС *Android* и каким образом обеспечивается безопасная среда их функционирования. Понимание этих вопросов позволяет вести более осознанную разработку приложений.

Невозможно создать осмысленное *приложение*, не изучив внутреннюю организацию, свойственную приложениям, работающим на определенной платформе. В данном курсе, очевидно, необходимо изучить структуру и основные компоненты приложений, разрабатываемых для работы на смартфонах под управлением ОС *Android*. От типа мобильного устройства внутренняя организация приложений не зависит, т. е. *Android*-приложения, разработанные для смартфонов вполне смогут выполняться и на планшетах. В данной лекции рассматривается *архитектура Android* приложений, основанная на идее многократного использования компонентов, которые являются основными строительными блоками. Подробно описываются основные компоненты, а также такие важные понятия для мобильных приложений, работающих под управлением *Android*, как *манифест* приложения и ресурсы.

§8. Основные виды Android-приложений

Приступая к разработке мобильных приложений хорошо бы иметь *представление* о том, какие виды приложений существуют. Дело в том, что если удастся определить к какому типу относится *приложение*, то становится понятнее на какие моменты в процессе его разработки необходимо обращать основное внимание. Можно выделить следующие виды приложений:

- **Приложения переднего плана** выполняют свои функции только, когда видимы на экране, в противном же случае их выполнение приостанавливается. Такими приложениями являются, например, игры, текстовые редакторы, видеопроигрыватели. При разработке таких приложений необходимо очень внимательно изучить жизненный цикл активности, чтобы переключения в фоновый режим и обратно проходили гладко (бесшовно), т. е. при возвращении приложения на передний план было незаметно, что оно вообще куда-то пропадало. Для достижения этой гладкости необходимо следить за тем, чтобы при входе в фоновый режим приложение сохраняло свое состояние, а при выходе на передний план восстанавливало его. Еще один важный момент, на который обязательно надо обратить внимание при разработке приложений переднего плана, удобный и интуитивно понятный интерфейс.
- **Фоновые приложения** после настройки не предполагают взаи-

модействия с пользователем, большую часть времени находятся и работают в скрытом состоянии. Примерами таких приложений могут служить, службы экранирования звонков, SMS-автоответчики. В большинстве своем фоновые приложения нацелены на отслеживание событий, порождаемых аппаратным обеспечением, системой или другими приложениями, работают незаметно. Можно создавать совершенно невидимые сервисы, но тогда они будут неуправляемыми. Минимум действий, которые необходимо позволить пользователю: санкционирование запуска сервиса, настройка, приостановка и прерывание его работы при необходимости.

- **Смешанные приложения** большую часть времени работают в фоновом режиме, однако допускают взаимодействие с пользователем и после настройки. Обычно взаимодействие с пользователем сводится к уведомлению о каких-либо событиях. Примерами таких приложений могут служить мультимедиа-проигрыватели, программы для обмена текстовыми сообщениями (чаты), почтовые клиенты. Возможность реагировать на пользовательский ввод и при этом не терять работоспособности в фоновом режиме является характерной особенностью смешанных приложений. Такие приложения обычно содержат как видимые активности, так и скрытые (фоновые) сервисы, и при взаимодействии с пользователем должны учитывать свое текущее состояние. Возможно потребуется обновлять

графический интерфейс, если приложение находится на переднем плане, или же посылать пользователю уведомления из фонового режима, чтобы держать его в курсе происходящего. И эти особенности необходимо учитывать при разработке подобных приложений.

- **Виджеты** - небольшие приложения, отображаемые в виде графического объекта на рабочем столе. Примерами могут служить, приложения для отображения динамической информации, такой как заряд батареи, прогноз погоды, дата и время. Разумеется, сложные приложения могут содержать элементы каждого из рассмотренных видов. Планируя разработку приложения, необходимо определить способ его использования, только после этого приступать к проектированию и непосредственно разработке.

§9. Безопасность

Обратим внимание на организацию исполнения приложений в ОС *Android*. Как уже было отмечено приложения под Android разрабатываются на языке программирования *Java*, компилируется в *файл* с расширением .apk, после этот *файл* используется для установки приложения на устройства, работающие под управлением *Android*. После установки каждое *Android* приложение "живет"в своей собственной безопасной "песочнице рассмотрим, как это выглядит:

- операционная система Android является многопользовательской ОС, в которой каждое приложение рассматривается как отдельный пользователь;
- по умолчанию, система назначает каждому приложению уникальный пользовательский ID, который используется только системой и неизвестен приложению;
- система устанавливает права доступа ко всем файлам приложения следующим образом: доступ к элементам приложения имеет только пользователь с соответствующим ID;
- каждому приложению соответствует отдельный Linux процесс, который запускается, как только это необходимо хотя бы одному компоненту приложения, процесс прекращает работу, когда ни один компонент приложения не использует его или же системе требуется

освободить память для других (возможно, более важных) приложений;

- каждому процессу соответствует отдельный экземпляр виртуальной машины Dalvik, в связи с этим код приложения исполняется изолировано от других приложений.

Перечисленные идеи функционирования приложения в ОС *Android* реализуют принцип минимальных привилегий, т. е. каждому приложению, по умолчанию, разрешен *доступ* только к компонентам, необходимым для его работы и никаким больше. Таким образом обеспечивается очень безопасная среда функционирования приложений.

Однако, в случае необходимости приложения могут получить *доступ* к данным других приложений и системным сервисам (услугам). В случае, когда двум приложениям необходимо иметь *доступ* к файлам друг друга, им присваивается один и тот же пользовательский *ID*. Для экономии системных ресурсов такие приложения запускаются в одном Linux процессе и делят между собой один и тот же экземпляр виртуальной машины, в этом случае приложения также должны быть подписаны одним сертификатом. В случае же, когда приложению требуется *доступ* к системным данным, например, контактам, *SMS* сообщениям, картам памяти, камере, Bluetooth и т. д., пользователю необходимо дать приложению такие полномочия во время установки его на устройство.

§10. Архитектура приложения, основные компоненты

Вот и пришла пора поговорить непосредственно о внутренней организации приложений под *Android*: обсудить их архитектуру и основные компоненты.

Архитектура Android приложений основана на идее многократного использования компонентов, которые являются основными строительными блоками. Каждый *компонент* является отдельной сущностью и помогает определить общее *поведение приложения*.

Система *Android* выстроена таким образом, что любое *приложение* может запускать необходимый *компонент* другого приложения. Например, если *приложение* предполагает использование камеры для создания фотографий, совершенно необязательно создавать в этом приложении *активность* для работы с камерой. Наверняка на устройстве уже есть *приложение* для получения фотографий с камеры, достаточно запустить соответствующую *активность*, сделать фотографию и вернуть ее в *приложение*, так что *пользователь* будет считать, что камера часть приложения, с которым он работает.

Когда система запускает *компонент*, она запускает процесс приложения, которому принадлежит *компонент*, если он еще не запущен, и создает экземпляры классов, необходимых компоненту. Поэтому в отличие от большинства других систем, в системе *Android* приложения не имеют единой точки входа (нет метода main(), например). В силу запус-

ка каждого приложения в отдельном процессе и ограничений на *доступ* к файлам, *приложение* не может напрямую активировать *компонент* другого приложения. Таким образом для активации компонента другого приложения необходимо послать системе сообщение о намерении запустить определенный *компонент*, система активирует его.

Можно выделить четыре различных типа компонентов, каждый тип служит для достижения определенной цели и имеет свой особый *жизненный цикл*, который определяет способы создания и разрушения соответствующего компонента. Рассмотрим основные компоненты *Android*-приложений.

Активности (Activities). *Активность* - это видимая часть приложения (экран, окно, форма), отвечает за *отображение* графического интерфейса пользователя. При этом *приложение* может иметь несколько активностей, например, в приложении, предназначенном для работы с электронной почтой, одна *активность* может использоваться для отображения списка новых писем, другая *активность* - для написания, и еще одна - для чтения писем. Несмотря на то, что для пользователя *приложение* представляется единым целым, все активности приложения не зависят друг от друга. В связи с этим любая из этих активностей может быть запущена из другого приложения, имеющего *доступ* к активностям данного приложения. Например, *приложение* камеры может запустить *активность*, создающую новые письма, чтобы отправить только что сделанную фотографию адресату, указанному пользователем.

Сервисы (Services). Сервис - *компонент*, который работает в фоновом режиме, выполняет длительные по времени *операции* или работу для удаленных процессов. Сервис не предоставляет пользовательского интерфейса. Например, сервис может проигрывать музыку в фоновом режиме, пока *пользователь* использует другое *приложение*, может загружать данные из сети, не блокируя взаимодействие пользователя с активностью. Сервис может быть запущен другим компонентом и после этого работать самостоятельно, а может остаться связанным с этим компонентом и взаимодействовать с ним.

Контент-провайдеры (Content providers). Контент-провайдер управляет распределенным множеством данных приложения. Данные могут храниться в файловой системе, в базе данных SQLite, в сети, в любом другом доступном для приложения месте. Контент-провайдер позволяет другим приложениям при наличии у них соответствующих прав делать запросы или даже менять данные. Например, в системе *Android* есть контент-провайдер, который управляет информацией о контактах пользователя. В связи с этим, любое *приложение* с соответствующими правами может сделать *запрос* на чтение и запись информации какого-либо контакта. Контент-провайдер может быть также полезен для чтения и записи приватных данных приложения, не предназначенных для доступа извне.

Приемники широковещательных сообщений (Broadcast Receivers). Приемник - *компонент*, который реагирует на широкове-

щательные извещения. Большинство таких извещений порождаются системой, например, извещение о том, что экран отключился или низкий заряд батареи. Приложения также могут инициировать *широковещание*, например, разослать другим приложениям сообщение о том, что некоторые данные загружены и доступны для использования. Хотя приемники не отображают пользовательского интерфейса, они могут создавать уведомление на панели состояний, чтобы предупредить пользователя о появлении сообщения. Такой приемник служит проводником к другим компонентам и предназначен для выполнения небольшого объема *работ*, например, он может запустить соответствующий событию сервис.

Все рассмотренные компоненты являются наследниками классов, определенных в *Android SDK*.

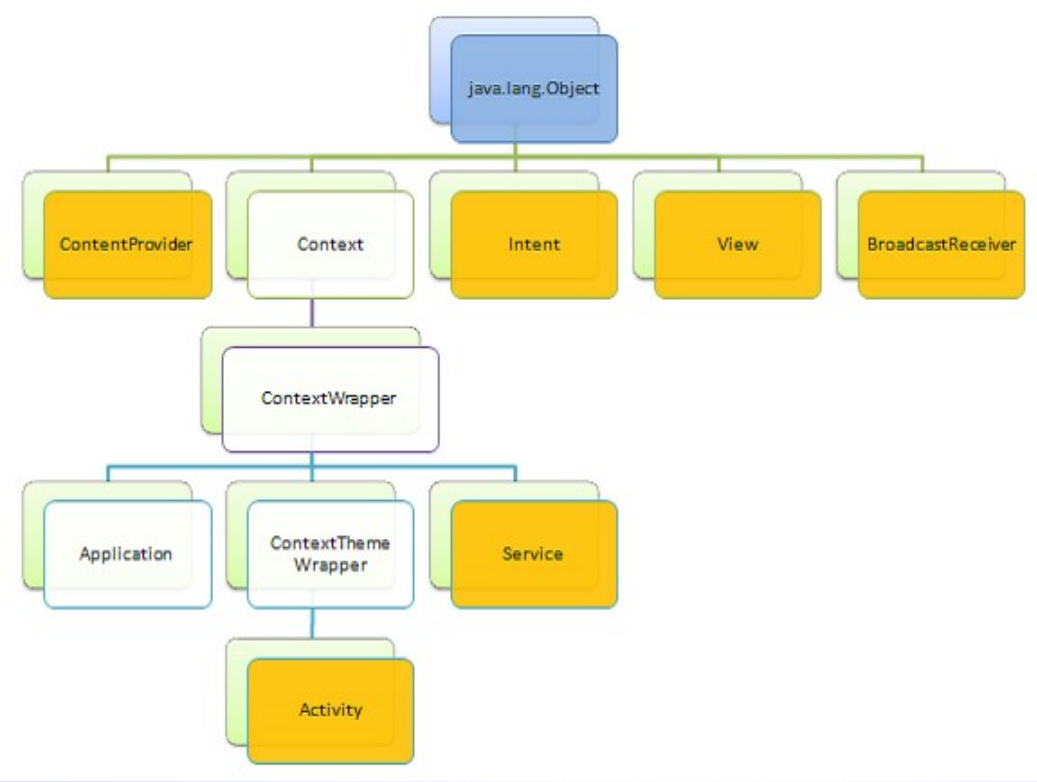


Рисунок 1. Иерархия классов Android SDK

(источник: здесь)

На рис. 1 показана *иерархия* основных классов *Android SDK*, с которыми обычно имеет дело разработчик. На самом деле классов намного больше, желтым цветом выделены классы, с которыми разработчик ра-

ботает непосредственно, наследует от них свои классы. Остальные классы не менее важны, но они реже используются напрямую. Для начала рассмотрим классы Intent и View.

Класс View является основным строительным блоком для компонентов пользовательского интерфейса (UI), он определяет прямоугольную область экрана и отвечает за прорисовку и обработку событий. Является базовым классом для виджетов (GUI widgets), которые используются для создания интерактивных компонентов пользовательского интерфейса: кнопок, текстовых полей и т. д. А также является базовым классом для класса ViewGroup, который является невидимым контейнером для других контейнеров и виджетов, определяет свойства расположения компонентов пользовательского интерфейса. *Интерфейс Android*-приложения представляет собой иерархию UI компонентов (см. рис. 2), можно описать эту иерархию программно, но более простым и эффективным способом задать расположение элементов интерфейса является *XML файл*, который предоставляет удобную для восприятия структуру компоновки (*layout* file). Во время исполнения *XML файл* автоматически превращается в *дерево* соответствующих объектов. Подробнее о классе View, свойствах и методах: здесь.

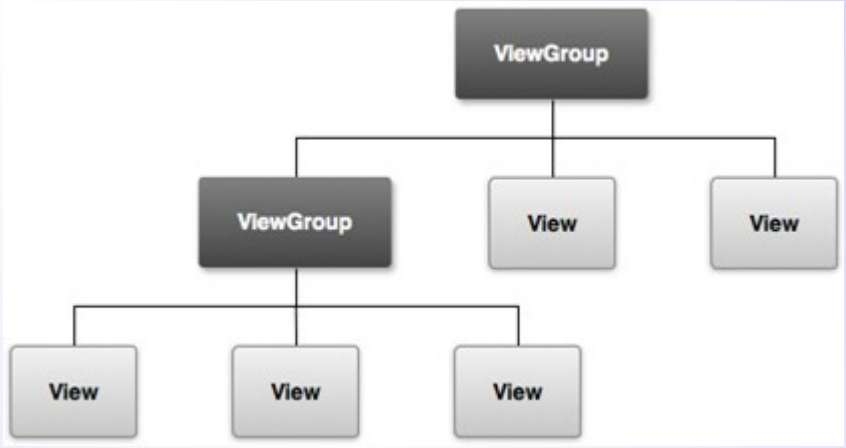


Рисунок 2. Иерархия компонентов, определяющая компоновку интерфейса пользователя

Объекты-экземпляры класса Intent используются для передачи сообщений между основными компонентами приложений. Известно, что три из четырех основных компонентов: активности, сервисы и приемники широковещательных сообщений, могут быть активированы с помощью сообщений, которые называются намерениями. Такие сообщения являются инструментом позднего связывания компонентов одного или нескольких приложений. Экземпляр класса Intent представляет собой структуру данных, содержащую описание *операции*, которая должна быть выполнена, и обычно используется для запуска активности или сервиса. В случае с приемниками широковещательных сообщений *объ-*

ект Intent содержит описание события, которое произошло или было объявлено.

Для каждого типа компонентов существуют свои *механизмы* передачи намерений.

- Чтобы запустить активность или вызвать у работающей активности новое действие, необходимо передать объект-намерение в метод Context.startActivity() или Activity.startActivityForResult().
- Чтобы запустить сервис или доставить новые инструкции работающему сервису, необходимо передать объект-намерение в метод Context.startService(). Также объект-намерение может быть передан в метод Context.bindService(), чтобы связать между собой вызывающий компонент и сервис.
- Чтобы доставить объект-намерение всем заинтересованным приемникам широковещательных сообщений, необходимо передать его в любой из широковещательных методов: Context.sendOrderedBroadcast(), Context.sendStickyBroadcast(), Context.sendBroadcast().

В каждом случае система *Android* в ответ на намерение находит соответствующий *компонент*: *активность*, сервис или множество широковещательных приемников и запускает его если необходимо. В этой системе сообщений не случается накладок: сообщение-намерение, отправ-

ленное определенному компоненту, будет получено именно этим компонентом и никем другим.

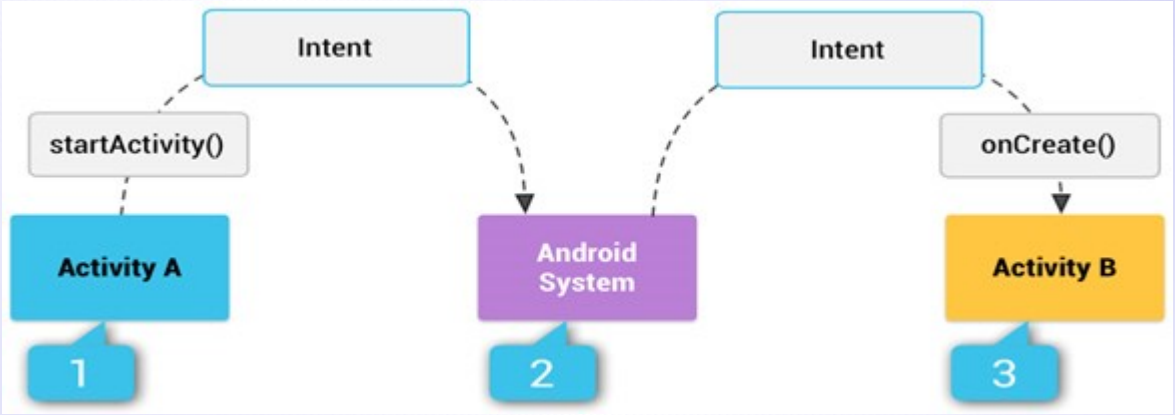


Рисунок 3. Передача намерений (Intent)

На рис. 3 можно увидеть как происходит передача намерений (Intent), в данном случае одна *активность* запускает другую. [6] *Активность* A создает намерение (Intent) с описанием действия и передает его в метод startActivity(). [7] Система *Android* проверяет все приложения на совпадение с намерением, когда совпадение найдено, [8] система запускает соответствующую *активность*, для чего вызывает метод onCreate() и передает в него *объект*-намерение Intent. Подробнее о классе Intent: здесь и здесь.

Пришло время более серьезно рассмотреть основные компоненты: активности, сервисы, контент-провайдеры, приемники широковещательных сообщений. В первую *очередь* нас будет интересовать *жизненный цикл* этих компонентов. Что же такое этот *жизненный цикл*? *Жизненный цикл* можно рассматривать, как процесс функционирования компонента: начиная с момента создания и запуска, включая активный и неактивный периоды работы, и, заканчивая уничтожением и освобождением ресурсов.

10.1 Активности (Activities)

Активность - окно, несущее графический интерфейс пользователя. Окно активности обычно занимает весь экран устройства, однако вполне возможно создавать полупрозрачные или плавающие диалоговые окна. Мобильные приложения обычно являются многооконными, т. е. содержат несколько активностей, по одной на каждое окно. Одна из активностей определяется как "главная и именно ее пользователь видит при первом запуске приложения.

Каждый экран приложения является наследником класса Activity. Для создания активности необходимо создать класс-наследник класса Activity напрямую или через любого его потомка. В этом классе необходимо реализовать все методы, вызываемые системой для управления жизненным циклом активности. Таких методов семь:

onCreate() - метод, вызываемый системой при создании активности. В реализации метода необходимо инициализировать основные компоненты активности и в большинстве случаев вызвать метод setContentView() для подключения соответствующего XML-файла компоновки (layout file). После метода onCreate() всегда вызывается метод onStart().

onRestart() - метод, вызываемый системой при необходимости запустить приостановленную *активность*. После этого метода всегда вызывается метод onStart().

onStart() - метод, вызываемый системой непосредственно перед тем, как активность станет видимой для пользователя. После этого метода вызывается onResume().

onResume() - метод, вызываемый системой непосредственно перед тем, как активность начнет взаимодействовать с пользователем. После этого метода всегда вызывается onPause().

onPause() - метод, вызываемый системой при потере активностью фокуса. В этом методе необходимо фиксировать все изменения, которые должны быть сохранены за пределами текущей сессии. После этого метода вызывается onResume(), если *активность* вернется на передний план, или onStop(), если активность будет скрыта от пользователя.

onStop() - метод, вызываемый системой, когда активность становится невидимой для пользователя. После этого метода вызывается либо onRestart(), если активность возвращается к взаимодействию с пользователем, либо onDestroy(), если активность уничтожается.

onDestroy() - метод, вызываемый системой перед уничтожением активности. Этот метод вызывается либо когда активность завершается, либо когда система уничтожает активность, чтобы освободить ресурсы. Можно различать эти два сценария с помощью метода isFinishing(). Это последний вызов, который может принять активность.

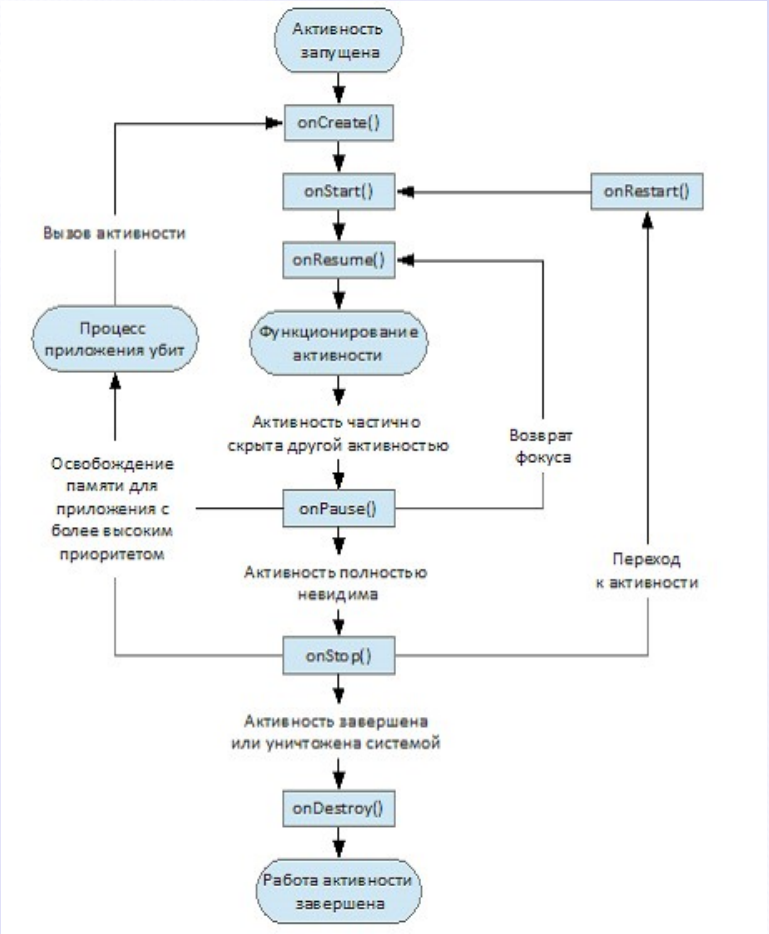


Рисунок 4. Жизненный цикл активности

(источник: здесь).

При реализации вышеперечисленных методов первым делом всегда необходимо вызывать соответствующий метод предка.

Рассмотренные методы определяют жизненный цикл *активности*. На рис. 4 можно увидеть пути, по которым активность может переходить из одного состояния в другое. В прямоугольниках указаны методы, которые вызываются при смене состояний активности.

Фактически активность может существовать в одном из трех состояний:

- **Выполняется (running)**. Активность находится на переднем плане и удерживает фокус ввода. Если внимательно рассмотреть (рис. 4) можно заметить, что в это состояние активность попадает после вызова метода onResume(). Пока активность находится в этом состоянии ее процесс не может быть уничтожен системой.
- **Приостановлена**. Активность частично видима, однако фокус ввода потерян. В это состояние активность попадает после вызова метода onPause() (рис. 4). В этом состоянии активность поддерживается в "боевой готовности т.е. в любой момент может получить фокус ввода и стать активной. Однако в этом состоянии процесс активности может быть уничтожен системой, в случае экстремальной нехватки памяти.
- **Остановлена**. Активность полностью невидима. В это состояние

активность попадает после вызова метода onStop() (рис. 4). В этом состоянии *активность* может быть "вызвана к жизни она сохраняет все состояния и необходимую для восстановления информацию, однако процесс активности может быть уничтожен, если память понадобится для других целей.

10.2 Сервисы (Services)

Сервис (Service) является компонентом приложения, предназначенным для выполнения длительных операций в фоновом режиме. Существует два способа существования сервисов:

- первый заключается в том, что сервис запущен (started) и работает самостоятельно в фоновом режиме, так он может работать неопределенно долго, пока не выполнит свою задачу;
- второй заключается в том, что сервис привязан (bound) к некоторому компоненту или нескольким компонентам, в этом случае сервис предлагает интерфейс для взаимодействия с компонентом и работает пока привязан хотя бы к одному компоненту, как только связь со всеми компонентами разрывается сервис завершает свою работу.

Для создания сервиса необходимо создать класс-наследник класса Service напрямую или через любого его потомка. При этом в реализации класса необходимо переопределить (т. е. написать свою реализа-

цию) некоторые методы, управляющие ключевыми аспектами жизненного цикла *сервиса* и обеспечивающие механизм связывания компонентов с сервисом, в соответствующем случае. Рассмотрим наиболее важные методы требующие реализации при создании сервиса.

onStartCommand() - метод, вызываемый системой, когда некоторый компонент, например *активность*, вызывает метод startService(). В этом случае сервис запускается и может работать в фоновом режиме неопределенно долго, поэтому необходимо позаботиться об остановке сервиса, когда он выполнит свою работу. Для остановки сервиса используется метод stopSelf() в случае, когда сервис сам прекращает свою работу, или stopService() в случае, когда работу сервиса прекращает некоторый компонент. Нет необходимости писать реализацию метода onStartCommand(), если не предполагается самостоятельной работы сервиса (т. е. он будет работать только в связке с некоторыми компонентами).

onBind() - метод, вызываемый системой, когда некоторый компонент желает привязать к себе *сервис* и вызывает метод bindService(). Этот метод должен возвращать реализацию интерфейса IBinder, которая может быть использована компонентом-клиентом для взаимодействия с сервисом. Метод onBind() необходимо реализовать в любом случае, но, если не предполагается связывания сервиса с какими-либо компонентами, возвращаемое значение должно быть равным null.

Необходимо отметить, что сервис может быть запущен как самостоятельная единица, а в последствии может быть привязан к некоторым компонентам. В этом случае в сервисе должны быть обязательно реализованы оба метода onStartCommand() и onBind().

onCreate() - метод, вызываемый системой, при первом обращении к сервису для выполнения первоначальных настроек. Этот метод вызывается до вызова методов onStartCommand() и/или onBind().

onDestroy() - метод, вызываемый системой, когда *сервис* либо выполнил все действия, для которых создавался, либо больше не связан ни с одним компонентом, т. е. его услуги больше не требуются. В реализации этого метода необходимо предусмотреть освобождение всех ресурсов, таких как потоки, зарегистрированные слушатели, приемники и т. д. Вызов этого метода является последними вызовом, который может получить сервис.

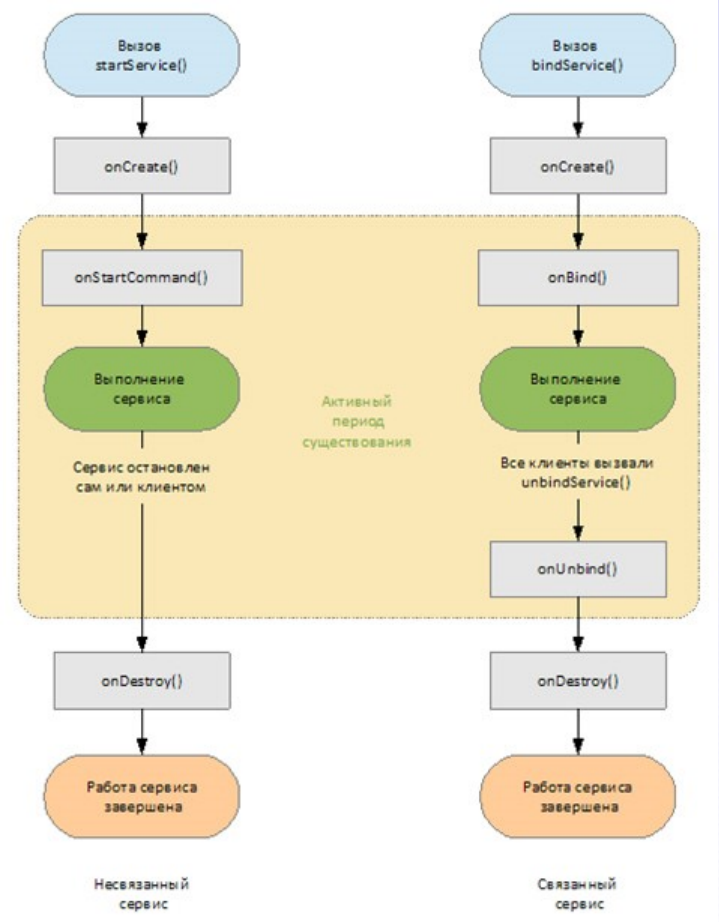


Рисунок 5. Жизненный цикл сервиса

(источник: здесь).

На рис. 5 показан жизненный цикл *сервиса*, левая диаграмма показывает жизненный цикл самостоятельного сервиса, правая - жизненный цикл сервиса, привязанного к некоторым компонентам. На рисунке хорошо видно, что жизненный цикл сервиса намного проще жизненного цикла *активности*. Однако для разработчика понимание того, как именно сервис создается, запускается и завершает свою работу, может оказаться даже более важным, т. к. сервис работает в фоновом режиме и пользователь может и не осознавать, что в некоторых случаях он имеет дело с работой сервисов.

Android принудительно останавливает работу сервисов только, когда ресурсов системы не хватает для активности, которая работает в данный момент на переднем плане. Приоритет работающих сервисов всегда выше, чем у приостановленных или полностью невидимых активностей, а если сервис привязан к выполняющейся активности, то его приоритет еще выше. С другой стороны, со временем приоритет самостоятельно работающего сервиса понижается и его шансы быть принудительно остановленным системой в случае нехватки ресурсов повышаются. В связи с этим имеет смысл проектировать сервис таким образом, чтобы через некоторое время он требовал у системы перезапуска. В случае если система все таки экстренно завершила работу сервиса, она перезапустит его как только освободятся ресурсы.

Подробнее о создании, использовании и удалении сервисов: здесь и

здесь.

10.3 Контент-провайдеры (Content Providers)

Контент-провайдер управляет доступом к хранилищу данных. Для реализации провайдера в Android приложении должен быть создан набор классов в соответствии с манифестом приложения. Один из этих классов должен быть наследником класса ContentProvider, который обеспечивает интерфейс между контент-провайдером и другими приложениями. Основное назначение этого компонента приложения заключается в предоставлении другим приложениям доступа к данным, однако ничто не мешает в приложении иметь *активность*, которая позволит пользователю запрашивать и изменять данные, находящиеся под управлением контент-провайдера.

В мобильных приложениях контент-провайдеры необходимы в следующих случаях:

- приложение предоставляет сложные данные или файлы другим приложениям;
- приложение позволяет пользователям копировать сложные данные в другие приложения;
- приложение предоставляет специальные варианты поиска, используя поисковую платформу (framework).

Если приложение требует использования *контент-провайдера*, необходимо выполнить несколько этапов для создания этого компонента:

1. **Проектирование способа хранения данных**. Данные, с которыми работают контент-провайдеры, могут быть организованы двумя способами:

   - Данные представлены файлом, например, фотографии, аудио или видео. В этом случае необходимо хранить данные в собственной области памяти приложения. В ответ на запрос от другого приложения, провайдер может возвращать ссылку на файл.
   - Данные представлены некоторой структурой, например, таблица, массив. В этом случае необходимо хранить данные в табличной форме. Строка таблицы представляет собой некоторую сущность, например, сотрудник или товар. А столбец - некоторое свойство этой сущности, например, имя сотрудника или цена товара. В системе Android общий способ хранения подобных данных - база данных SQLite, но можно использовать любой способ постоянного хранения.

   Больше о хранении данных в Android можно узнать по ссылке: здесь.

2. **Создание класса-наследника от класса** ContentProvider на-

прямую или через любого его потомка. При этом в реализации класса необходимо переопределить (т. е. написать свою реализацию) обязательные методы.

query() - метод, извлекающий данные из провайдера, в качестве аргументов получает таблицу, строки и столбцы, а также порядок сортировки результата, возвращает объект типа Cursor.

insert() - метод, добавляющий новую строку, в качестве аргументов получает таблицу, и значения элементов строки, возвращает URI добавленной строки.

update() - метод, обновляющий существующие строки, в качестве аргументов получает таблицу, строки для обновления и новые значения элементов строк, возвращает количество обновленных строк.

delete() - метод, удаляющий строки, в качестве аргументов принимает таблицу и строки для удаления, возвращает количество удаленных строк.

getType() - метод, возвращающий String в формате MIME, который описывает тип данных, соответствующий URI. Подробнее: здесь.

onCreate() - метод, вызываемый системой, сразу после создания провайдера, включает инициализацию провайдера. Стоит отметить, что провайдер не создается до тех пор, пока объект ContentResolver не попытается получить к нему доступ.

Созданный *контент-провайдер* управляет доступом к структурированным данным, выполняя обработку запросов от других приложений. Все запросы, в конечном итоге, вызывают объект ContentResolver, который в свою очередь вызывает подходящий метод объекта ContentProvider для получения доступа. Все вышеперечисленные методы, кроме onCreate(), вызываются приложением-клиентом. И все эти методы имеют такую же сигнатуру, как одноименные методы класса ContentResolver.

Подробнее о классе ContentProvider: здесь.

3. **Определение строки авторизации провайдера, URI для его строк и имен столбцов**. Если от провайдера требуется управление намерениями, необходимо определить действия намерений, внешние данные и флаги. Также необходимо определить разрешения, которые необходимы приложениям для доступа к данным провайдера. Все эти значения необходимо определить как константы в отдельном классе, этот класс в последствии можно предоставить другим разработчикам.

Подробнее об URI: здесь. Подробнее о намерениях: здесь.

10.4 Приемники широковещательных сообщений (Broadcast Receivers)

Каждый широковещательный приемник является наследником класса BroadcastReceiver. Этот класс рассчитан на получение объектов-намерений отправленных методом sendBroadcast().

Можно выделить две разновидности широковещательных сообщений:

- **Нормальные широковещательные сообщения** передаются с помощью Context.sendBroadcast в асинхронном режиме. Все приемники срабатывают в неопределенном порядке, часто в одно и то же время.
- **Направленные широковещательные сообщения** передаются с помощью Context.sendOrderedBroadcast только одному приемнику в один момент времени. Как только приемник сработает, он может передать сообщение следующему приемнику, а может прервать вещание так, что больше ни один приемник это сообщение не получит.

Даже в случае нормального широковещания могут сложиться ситуации, в которых система будет передавать сообщения только одному приемнику в один момент времени. Особенно это актуально для приемников, которые требуют создания процессов, чтобы не перегружать

систему новыми процессами. Однако в этом случае ни один приемник не может прервать широковещание.

Объект типа BroadcastReceiver действителен только во время вызова метода onRecieve(), как только метод выполнен, система завершает работу объекта и больше не активирует его.

Подробнее о приемниках широковещательных сообщений: здесь.

§11. Манифест приложения

Корневой каталог каждого приложения под *Android* должен содержать *файл* AndroidManifest.*xml* (в точности с таким названием).
Манифест приложения содержит всю необходимую информацию, используемую системой для запуска и выполнения приложения. Основная *информация*, содержащаяся в манифесте:

- Имя Java пакета приложения, которое используется как уникальный идентификатор приложения.
- Описание компонентов приложения: *активностей*, *сервисов*, приемников широковещательных сообщений и *контент-провайдеров*, которые составляют приложение. Для каждого компонента приложения определено имя соответствующего класса и объявлены их основные свойства (например, с какими сообщениями-намерениями они могут работать). Эта информация позволяет системе Android узнать какие компоненты и при каких условиях могут быть запущены.
- Определение процессов, в которых будут выполняться компоненты приложения.
- Объявление полномочий, которыми должно обладать приложение для доступа к защищенным частям API и взаимодействия с другими приложениями.

- Объявление полномочий, которыми должны обладать другие приложения для взаимодействия с компонентами данного.
- Список вспомогательных классов, которые предоставляют информацию о ходе выполнения приложения. Эти объявления содержатся в *манифесте* пока идет разработка и отладка приложения, перед публикацией приложения они удаляются.
- Определение минимального уровня Android API для приложения.
- Список библиотек связанных с приложением

В файле манифеста только два элемента: <manifest> и <application> являются обязательными и при этом встречаются ровно по одному разу. Остальные элементы могут встречаться несколько раз или не появляться совсем, в этом случае *манифест* определяет пустое *приложение*.

Следующий листинг демонстрирует общую структуру файла манифеста.

```
<?xml version="1.0"encoding="utf-8"?>
<manifest>
<uses-permission />
<permission />
<permission-tree />
<permission-group />
<instrumentation />
<uses-sdk />
```

<uses-configuration />
<uses-feature />
<support-screens />
<compatible-screens />
<supports-gl-texture />
<application>
<activity>
<intent-filter>
<action />
<category />
<data />
</intent-filter>
<meta-data />
</activity>
<activity-alias>
<intent-filter> ... </intent-filter>
<meta-data />
</activity-alias>
<service>
<intent-filter> ... </intent-filter>
<meta-data />
</service>
<receiver>

```
<intent-filter> ... </intent-filter>
<meta-data />
</receiver>
<provider>
<grant-uri-permission />
<meta-data />
<path-permission />
</provider>
<uses-library />
</application>
</manifest>
```

Листинг 3.1. Структура файла AndroidManifest.xml

В *манифесте* элементы одного уровня, такие как <activity>, <service>, <receiver>, <provider>, могут следовать друг за другом в любой последовательности. Элемент <activity-alias> является исключением из этого правила, он должен следовать за соответствующей *активностью*.

Более предметно разговор о файле манифеста и его основных элементах пойдет в лабораторных работах.

Подробности: здесь.

§12. Ресурсы

При разработке мобильных приложений необходимо выработать привычку отделять ресурсы приложения от кода. К ресурсам приложения могут относиться: изображения, строки, цвета, компоновки элементов пользовательского интерфейса (*layout*) и т. д. Отделение ресурсов от кода позволяет использовать *альтернативные* ресурсы для различных конфигураций устройств: язык, разрешение экрана и т. д. Для обеспечения совместимости с различными конфигурациями, ресурсы необходимо сгруппировать в директории по типу ресурсов и конфигурации устройства, полученные директории поместить в папку **res**/.

Для любого типа ресурсов можно определить две группы. Первая определяет ресурсы, которые будут использоваться независимо от конфигурации устройства или в том случае, когда под конфигурацию нет подходящих альтернативных ресурсов. Эта *группа* называется ресурсы по умолчанию (*default*). Вторая *группа* определяет ресурсы, подходящие для определенной конфигурации устройства, размещается в директории с названием, обозначающим данную конфигурацию. Такие ресурсы называются альтернативными.

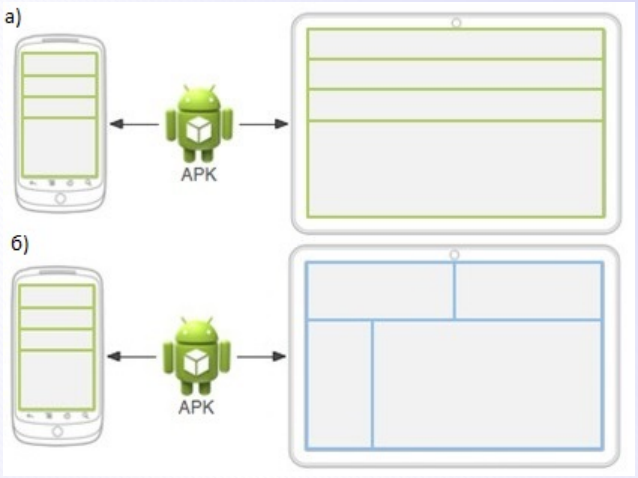


Рисунок 1. Использование ресурсов: а) используется компоновка по умолчанию (приложение не содержит альтернативы); б) каждое устройство использует соответствующую компоновку

Каждый тип ресурсов необходимо размещать в специальной поддиректории папки **res**/. Рассмотрим основные из этих поддиректорий:

animator/ - содержит XML файлы, которые определяют свойства анимации;

anim/ - содержит XML файлы, которые определяют анимацию преобразований;

color/ - содержит XML файлы, которые определяют списки цветов;

drawable/ - содержит графические файлы или XML файлы, которые компилируются в графические ресурсы;

layout/ - содержит XML файлы, которые определяют компоновку элементов пользовательского интерфейса;

menu/ - содержит XML файлы, которые определяют все меню приложения;

values/ - содержит XML файлы, которые определяют простые значения, таких ресурсов как, строки, числа, цвета.

Следует отметить, что *файлы ресурсов* нельзя размещать в папку **res/** напрямую, они обязательно должны размещаться в соответствующем каталоге, иначе будет выдана ошибка компиляции.

Все ресурсы, которые содержатся в рассмотренных поддиректориях являются ресурсами по умолчанию. Понятно, что различные типы устройств могут требовать различных типов ресурсов. Например, для устройств с разными размерами экрана компоновки элементов пользовательского интерфейса должны отличаться. Рис. 1 показывает варианты внешнего вида приложения с использованием только компоновки по умолчанию (а) и с использованием альтернативных компоновок (б). Даже на схеме понятно, что при правильном *подходе* приложение, изменяющее свой внешний вид в зависимости от размера экрана привлекательнее, чем остающееся неизменным.

Чтобы определить зависимые от конфигурации альтернативы для *множества* ресурсов:

1. необходимо создать директорию в каталоге **res**/, присвоить этой директории имя в следующей форме:
   имя_ресурса-спецификатор_конфигурации, где
   - имя_ресурса - имя директории, соответствующего ресурса по умолчанию (см. выше);
   - спецификатор_конфигурации - имя, определяющее конфигурацию, для которой используются данные ресурсы. Полный список доступных спецификаторов: здесь;
2. необходимо сохранить ресурсы в новой директории, файл ресурсов должен называться в точности так же, как соответствующий файл ресурсов по умолчанию.

Например, если *компоновка* элементов пользовательского интерфейса сохранена, как ресурс по умолчанию, в папке **res/layout**/, можно (скорее даже нужно) определить альтернативную компоновку элементов пользовательского интерфейса, соответствующую горизонтальной (альбомной) ориентации экрана смартфона и сохранить ее в папке **res/layout-land**/. *Android* автоматически определит подходящую компоновку, сверяя текущее состояние устройства с именами папок в каталоге /**res**.

Все ресурсы после определения могут быть доступны по ссылке на их *ID*, которые определены в автоматически генерируемом классе R. Для каждого типа ресурсов в R классе существует *подкласс*, например, R.drawable для всех графических ресурсов. *ID* ресурса всегда имеет две составляющие:

- тип ресурса - все ресурсы группируются по типам, например, string, drawable, layout;
- имя ресурса - либо имя файла без расширения, либо значение атрибута android:name в XML файле для простого значения.

Получить *доступ* к ресурсу можно двумя способами:

- в коде: можно использовать выражения вида R.тип_ресурса.имя_ресурса, например, R.string.hello;
- в XML: используется специальный XML синтаксис, который соответствует ID определенному в R классе, например, @string/hello.

Более предметно разговор об использовании ресурсов в лабораторных работах.

Подробности: здесь.

§13. Основы разработки интерфейсов мобильных приложений

Аннотация: Большинство современных мобильных устройств имеют сенсорные дисплеи. Между традиционным оконным и тачевым интерфейсами существует огромная разница. Разработка удобного интерфейса для мобильных приложений является довольно сложной проблемой. Основной целью лекции является рассмотрение основ разработки интерфейсов мобильных приложений. В лекции рассказывается об особенностях визуального дизайна интерфейсов, строительных блоках и элементах управления. Приведены рекомендации по проектированию GUI под Android, а также имеется большое количество разнообразных примеров. В конце приведен список дополнительных источников. Описанные принципы помогут при разработке удобных пользовательских интерфейсов для мобильных приложений. Лекция может быть использована как часть курса или же отдельно от него для лучшего понимания особенностей интерфейса мобильных приложений.

Для подготовки данной лекции в качестве основного источника информации была использована книга "Алан Купер об интерфейсе"[11], однако задекларированные в ней принципы были переработаны в контексте программирования для мобильных устройств, а примеры заменены на более подходящие в рамках данного курса. Скриншоты приложений взяты из магазина приложений Google Play или сделаны са-

мостоятельно с использованием смартфона Мегафон SP-A20i Mint на платформе Intel Medfield.

Презентацию к данной лекции можно скачать здесь.

13.1 Визуальный дизайн интерфейсов

Силы, вложенные в разработку модели поведения программного продукта, будут потрачены впустую, если вы не сумеете должным образом донести до пользователей принципы этого поведения. В случае мобильных продуктов это делается визуальными средствами - путем отображения объектов на дисплее (в некоторых случаях целесообразно использовать тактильные ощущения от нажатия).

Визуальный *дизайн* интерфейсов - очень нужная и уникальная дисциплина, которую следует применять в сочетании с проектированием взаимодействия и промышленным дизайном. Она способна серьезно повлиять на эффективность и привлекательность продукта, но для полной реализации этого потенциала нужно не откладывать визуальный *дизайн* на потом, а сделать его одним из основных инструментов удовлетворения потребностей пользователей и бизнеса.

Изобразительное искусство, визуальный дизайн интерфейсов и прочие дисциплины дизайна

Художники и визуальные дизайнеры работают с одними и теми же изобразительными средствами, однако их деятельность служит различ-

ным целям. Цель художника - создать объект, взгляд на который вызывает эстетический отклик. Изобразительное искусство - способ самовыражения художника. Художник не связан почти никакими ограничениями. Чем необычнее и своеобразнее продукт его усилий, тем выше он ценится.

Дизайнеры создают объекты, которыми будут пользоваться другие люди. Если говорить о дизайнерах визуальных интерфейсов, то они ищут наилучшее представление, доносящее информацию о поведении программы, в проектировании которой они принимают участие. Придерживаясь целеориентированного подхода, они должны стремиться представлять поведение и информацию в понятном и полезном виде, который поддерживает маркетинговые цели организации и эмоциональные цели персонажей. Разумеется, визуальный дизайн пользовательских интерфейсов не исключает эстетических соображений, но такие соображения не должны выходить за рамки функционального каркаса.

Графический дизайн и пользовательские интерфейсы

Графические дизайнеры обычно очень хорошо разбираются в визуальных аспектах и хуже представляют себе понятия, лежащие в основе поведения программного продукта и взаимодействия с ним. Они способны создавать красивую и адекватную внешность интерфейсов, а кроме того - привносить фирменный стиль во внешний вид и поведение программного продукта. Для таких специалистов дизайн или проектирова-

ние интерфейса есть в первую очередь тон, стиль, композиция, которые являются атрибутами бренда, во вторую очередь - прозрачность и понятность информации и лишь затем - передача информации о поведении посредством ожидаемого назначения.

Дизайнерам визуальной части интерфейса необходимы некоторые навыки, которые присущи графическим дизайнерам, но они должны еще обладать глубоким пониманием и правильным восприятием роли поведения. Их усилия в значительной степени сосредоточены на организационных аспектах проектирования. В центре их внимания находится соответствие между визуальной структурой интерфейса с одной стороны и логической структурой пользовательской ментальной модели и поведения программы - с другой. Кроме того, их заботит вопрос о том, как сообщать пользователю о состояниях программы и что делать с когнитивными аспектами пользовательского восприятия функций.

Визуальный информационный дизайн

Информационные дизайнеры работают над визуализацией данных, содержимого и средств навигации. Усилия информационного дизайнера направлены на то, чтобы представить данные в форме, способствующей их верному истолкованию. Результат достигается через управление визуальной иерархией при помощи таких средств, как цвет, форма, расположение и масштаб. Распространенными объектами информационного

дизайна являются всевозможные графики, диаграммы и прочие способы отображения количественной информации.

Чтобы создавать привлекательные и удобные пользовательские интерфейсы, дизайнер интерфейсов должен владеть базовыми визуальными навыками - пониманием цвета, типографики, формы и композиции - и знать, как их можно эффективно применять для передачи поведения и представления информации, для создания настроения и стимулирования физиологических реакций. Дизайнеру интерфейса также требуется глубокое понимание принципов взаимодействия и идиом интерфейса, определяющих поведение продукта.

13.2 Строительные блоки визуального дизайна интерфейсов

Дизайн интерфейсов сводится к вопросу о том, как оформить и расположить визуальные элементы таким образом, чтобы внятно отразить поведение и представить информацию. Каждый элемент визуальной композиции имеет ряд свойств, и сочетание этих свойств придает элементу смысл. *Пользователь* получает возможность разобраться в интерфейсе благодаря различным способам приложения этих свойств к каждому из элементов интерфейса. В тех случаях, когда два объекта обладают общими свойствами, *пользователь* предположит, что эти объекты связаны или похожи. Когда пользователи видят, что свойства отличаются, они

предполагают, что объекты не связаны.

Создавая пользовательский *интерфейс*, проанализируйте перечисленные ниже визуальные свойства каждого элемента или группы элементов. Чтобы создать полезный и привлекательный пользовательский *интерфейс*, следует тщательно поработать с каждым из этих свойств.

Форма

Форма - главный признак сущности объекта для человека. Мы узнаем объекты по контурам. Если мы увидим на картинке синий ананас, мы его сразу опознаем, потому что мы помним его форму. И лишь потом мы удивимся странному цвету (см. рис. 1). При этом различение форм требует большей концентрация внимания, чем анализ цвета или размера. Поэтому форма - не лучшее свойство для создания контраста, если требуется привлечь внимание пользователя.

Рисунок 1. В первую очередь мы видим ананас, а уже потом начинаем задумываться, почему он синий

Размер

Более крупные элементы привлекают больше внимания, особенно если они значительно превосходят размерами окружающие элементы. Люди автоматически упорядочивают объекты по размеру и склонны оценивать их по размеру; если у нас есть текст в четырех размерах, предполагается, что относительная важность текста растет вместе с размером и что полужирный текст более важен, чем текст с нормальным начертанием. Таким образом, размер - полезное свойство для обозначения

информационных иерархий.

Цвет

Цветовые различия быстро привлекают внимание. В некоторых профессиональных областях цвета имеют конкретные значения, и этим можно пользоваться. Так, для бухгалтера красный цвет - отрицательные результаты, а черный - положительные.

Цвета приобретают смыслы и благодаря социальным контекстам, в которых проходит наше взросление. Например, белый цвет на Западе ассоциируется с чистотой и миром, а в Азии и арабских странах - с похоронами и смертью. При этом цвет изначально не обладает свойством упорядоченности и не выражается количественно, поэтому далеко не идеален для передачи информации такого рода. Кроме того, не следует делать цвет единственным способом передачи информации, поскольку цветовая слепота встречается довольно часто.

Применяйте цвет с умом. Чтобы создать эффективную визуальную систему, позволяющую пользователю выявлять сходства и различия объектов, используйте ограниченный набор цветов - эффект радуги перегружает восприятие пользователя и ограничивает возможности по передаче ему информации.

Выбор цветовой палитры для программы необходимо проводить очень осторожно. По разным данным, той или иной формой цветовой слепоты

страдают до 10% мужчин, и использование, например, красного и зеленого цветов для указания контраста затрудняет работу с приложением для этих людей.

Яркость

Понятия темного и светлого обретают смысл преимущественно в контексте яркости фона. На темном фоне темный текст почти не виден, тогда как на светлом он будет резко выделяться. Контрастность люди воспринимают легко и быстро, так что значение яркости может стать хорошим инструментом привлечения внимания к тем элементам, которые требуется подчеркнуть. Значение яркости - также упорядоченная переменная, например, более темные (с более низкой яркостью) цвета на карте легко интерпретируются: они обозначают большие глубины или большие значения других параметров.

Направление

Направление полезно, когда требуется передавать информацию об ориентации (вверх или вниз, вперед или назад). Помните, что восприятие направления может быть затруднено в случае некоторых форм и при малых размерах объектов, поэтому ее лучше использовать в качестве вторичного признака. Так, если требуется показать, что рынок акций пошел вниз, можно использовать направленную вниз стрелку красного цвета.

Текстура

Разумеется, изображенные на экране элементы не обладают настоящей текстурой, но способны создавать ее видимость. Текстура редко бывает полезна для передачи различий или привлечения внимания, поскольку требует значительной концентрации на деталях. И тем не менее текстура может быть важной подсказкой. Засечки и выпуклости на элементах пользовательского интерфейса обычно указывают, что элемент можно перетаскивать, а фаски или тени у кнопки усиливают ощущение, что ее можно нажать.

Расположение

Расположение - это переменная, упорядоченная и выражаемая количественно, а значит, полезная для передачи иерархии. Расположение также может служить средством создания пространственных отношений между объектами на экране и объектами реального мира (например, небо в верхней половине, земля в нижней).

Расположение элементов мобильного приложения очень сильно влияет на удобство использования и зависит от того, как пользователь будет держать устройство (см. рис. 2). Подробнее об этом будет рассказано в продолжении данного курса.

Рисунок 2. Различные варианты удержания смартфона

13.3 Элементы управления и дизайн навигации

Элементы управления - это доступные для манипулирования самодостаточные экранные объекты, посредством которых люди взаимодействуют с цифровыми продуктами. *Элементы управления* (controls, другое название - widgets - сокращение от *windows* gadgets - оконные приспособления) - это базовые строительные блоки графического пользовательского интерфейса. Рассматривая *элементы управления* с учетом целей пользователя, их можно разбить на четыре основные категории:

- **командные элементы управления**, применяемые для выполнения функций;
- **элементы выбора**, позволяющие выбирать данные или настройки;

- **элементы ввода**, применяемые для ввода данных;
- **элементы отображения**, используемые для наглядного непосредственного манипулирования.

Некоторые *элементы управления* сочетают в себе свойства более чем одной категории.

Командные элементы управления

Командные элементы управления выполняют действия, причем делают это немедленно. Главным и по сути единственным командным элементом является кнопка, которая обладает множеством вариантов отображения. Элементы меню также являются командными идиомами.

Кнопки

Кнопки обычно легко опознаются благодаря их псевдотрехмерности (рис. 3). Действие выполняется сразу после нажатия на кнопку. Часто особым образом выделяется кнопка по умолчанию, соответствующая наиболее часто используемому действию.

Кнопка - удобный и привлекательный с визуальной точки зрения элемент управления. Весь ее облик подсказывает, что на нее можно нажать, и это характеризует ее ожидаемое назначение. Рекомендуется изменять внешний вид нажатой кнопки, так как это облегчает понимание работы программы пользователем.

Рисунок 3. Скриншоты популярной игры "Cut the Rope". Кнопки кажутся выпуклыми, а нажатая кнопка меняет цвет

Кнопки-значки

Кнопки, помещенные на панель инструментов, обычно становятся квадратными, теряют текстовую надпись и обзаводятся пиктограммой - пояснением в виде графического значка (рис. 4).

Считается, что кнопки-значки очень удобны: они постоянно на виду и взаимодействовать с ними проще, чем с элементами раскрывающегося меню. Поскольку они постоянно видны, то легко запоминаются. У большинства пользователей не возникает вопросов относительно ожидаемого назначения кнопки. Проблема в том, что изображение на кнопке иногда

бывает непонятным. Например, пиктограмма с изображением дискеты, традиционно обозначающая сохранение, часто непонятна молодым пользователям, которые никогда не работали с реальными дискетами.

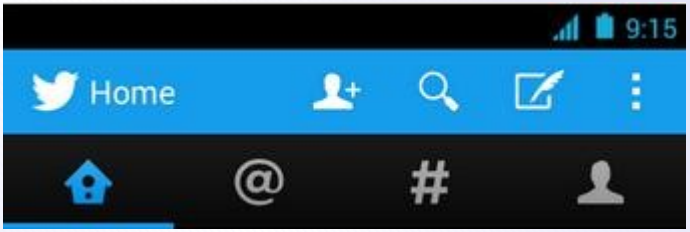


Рисунок 4. Кнопки-значки в Android-приложении Twitter

Гиперссылки

Текстовые гиперссылки - распространенный способ навигации в сети и веб-приложениях, однако при программировании для мобильных устройств их следует избегать. Дело в том, что попасть по ссылке пальцем с первого раза часто затруднительно и пользователей раздражает необходимость повторения этих действий.

Элементы управления выбором

Элементы выбора позволяют пользователю выбрать из группы допустимых объектов тот, с которым будет совершено действие. Элементы

выбора применяются также для действий по настройке. Распространенными элементами выбора являются флажки и списки.

Раньше использование элементов управления выбором не приводило к немедленному выполнению действий - требовалась еще и активация командного элемента. Сейчас возможны оба варианта. Если желательно дать пользователю возможность несколько раз осуществить выбор перед выполнением действия, следует создать явный командный элемент управления (кнопку). Если же пользователю полезно сразу видеть результат своих действий, и эти действия легко отменить, разумно сделать так, чтобы элемент выбора играл также и роль командного элемента.

Флажки

Назначение флажка очевидно. Щелкнув по флажку, пользователь немедленно увидит появившуюся галочку. Флажок прост, нагляден и изящен, однако основан на тексте. Качественный текст может исключить возможность неоднозначного толкования флажка. Однако этот же поясняющий текст вынуждает пользователя замедляться для прочтения, а также занимает значительное экранное пространство.

Традиционно флажки имеют квадратную форму. Не забывайте, что пользователи распознают визуальные объекты по форме, и квадратная форма флажков - важный стандарт.

Выключатели

Существует возможность сделать флажок более наглядным, применив в качестве основы кнопку-значок, которая может фиксироваться в нажатом состоянии. Такой элемент называется **выключателем** (рис. 5).

Рисунок 5. Один из самых распространенных выключателей - кнопка "Pause"в играх

Состояние выключателя остается неизменным до следующего щелчка. Выключатели экономно расходуют экранное пространство: они занимают меньше места, потому что их назначение описывается не с помощью текста, а посредством визуальных средств. Разумеется, это озна-

чает, что им присущ тот же недостаток обычных кнопок-значков - неоднозначность пиктограмм.

Триггеры

Кнопки-триггеры - это разновидность элементов управления. Они призваны экономить экранное пространство, к сожалению, ценой значительной дезориентации пользователя. Классический пример - размещение на одной кнопке функций воспроизведения и паузы для музыкального проигрывателя. Подводным камнем такого подхода является то, что элемент управления можно ошибочно посчитать индикатором состояния проигрывателя ("на паузе"или "идет воспроизведение"). Элемент управления может служить либо индикатором состояния, либо кнопкой переключения состояний, но не тем и другим одновременно (рис. 6).

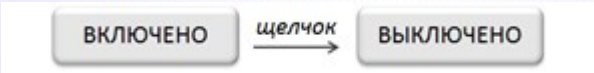


Рисунок 6. Если кнопка говорит "ВКЛЮЧЕНО когда находится в состоянии "выключено то непонятно, в каком же состоянии находится кнопка. Если кнопка говорит "ВЫКЛЮЧЕНО когда находится в состоянии "выключено тогда возникает вопрос, где искать кнопку "ВКЛЮЧЕНО"?

Радиокнопки

Радиокнопки внешне похожи на флажки (рис. 7), но являются взаимоисключающими, то есть выбор одного из вариантов автоматически аннулирует предыдущий выбор. В каждый момент времени может быть выбрана только одна кнопка. Радиокнопки всегда объединяются в группы из двух или более радиокнопок, причем в каждой группе одна радиокнопка всегда выбрана. Радиокнопки всегда круглые по той же причине, по которой флажки всегда имеют квадратную форму: именно такими они были изначально.

Радиокнопки занимают даже больше места, чем флажки, однако в некоторых случаях такой расход экранного пространства оправдан.

Кнопка-значок преобразовала радиокнопки так же, как флажки, заменив их в основном интерфейсе приложения. Если два или более выключателя объединены схемой взаимного исключения - так, чтобы в каждый момент мог быть включен лишь один из них, - они ведут себя точно так же, как радиокнопки. Так образуются радиокнопки со значками. Элементы управления цветом в Adobe Photoshop - хороший пример **радиокнопок со значками** (рис. 8).

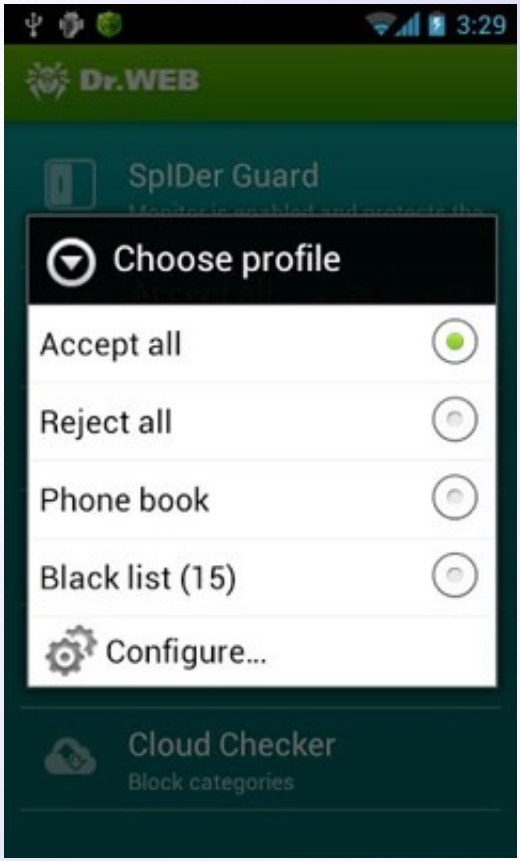


Рисунок 7. Радиокнопки Android-приложения Dr.Web

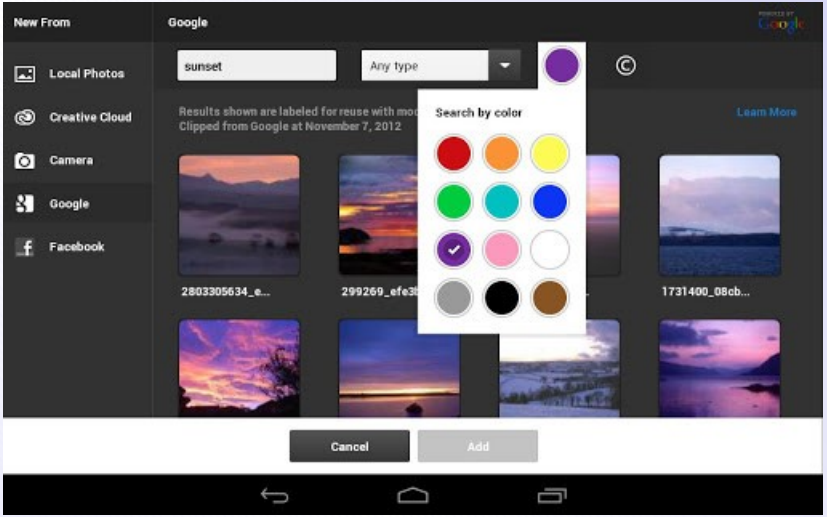


Рисунок 8. Элементы управления цветом в Android-приложении Adobe Photoshop представляют собой группу радиокнопок со значками

Списки

Элементы управления типа "список"позволяют осуществлять выбор из конечного множества текстовых строк, каждая из которых представляет команду, объект или признак. Подобно радиокнопкам, списки - мощный инструмент, упрощающий взаимодействие за счет устранения возможности неправильного выбора. Списки - это небольшие текстовые области с полосой прокрутки, автоматически подключаемой при необходимости (рис. 9). Пользователь может выбрать единственную строку текста, нажав на нее.

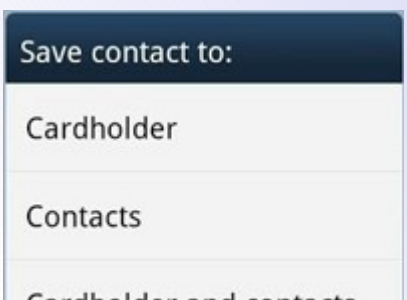


Рисунок 9. Стандартный список в Android

Раскрывающийся список - повсеместно встречающийся вариант обычного списка. Он показывает лишь выбранный элемент в одну строку, но если нажать на стрелку, открываются другие варианты выбора.

Элемент управления **представление в виде списка** предоставляет возможность сопровождать каждую строку текста пиктограммой. Такая возможность весьма полезна - существует множество ситуаций, когда можно упростить работу пользователя, располагая графические идентификаторы рядом со строками важных вариантов выбора (рис. 10).

Рисунок 10. Элемент управления типа "список"с пиктограммами в Android-приложении, позволяющий визуально оценивать погоду в различных городах

Комбо-списки и комбо-кнопки

Комбо-элементы представляют собой сочетание элементов. Комбо-кнопка - разновидность радиокнопки со значком. Обычно она выглядит как кнопка-значок с небольшой стрелкой, но если нажать на стрелку и удерживать ее в нажатом состоянии, разворачивается меню.

Комбо-список представляет собой сочетание списка и поля редактирования (рис. 11).

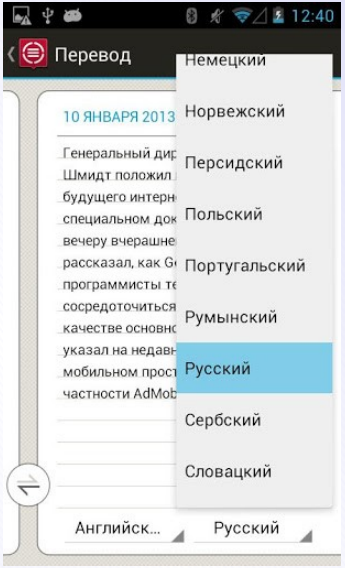


Рисунок 11. Элемент управления типа "список"с пиктограммами в Android-приложении, позволяющий визуально оценивать погоду в различных городах

Вариант с раскрывающимся списком значительно экономит экранное пространство. Комбо-список хорошо подходит для тех случаев, когда необходимо организовать выбор единственного объекта.

Элементы ввода

Элементы ввода дают пользователю возможность не только выбирать существующие сведения, но и вводить новую информацию. Самый

простой элемент - поле редактирования текста (поле ввода). В эту категорию попадают также такие элементы управления, как счетчики и ползунки.

Ограничивающие элементы ввода

Любой элемент управления, ограничивающий набор значений, доступных для ввода пользователем, является ограничивающим элементом ввода. Так, например, ползунок со шкалой значений от 0 до 100 является ограничивающим элементом ввода. Независимо от действий пользователя не может быть введено число, выходящее за диапазон определенных программой значений. Проще говоря, ограничивающие элементы ввода должны использоваться везде, где необходимо ограничить множество допустимых значений.

Ограничивающий элемент ввода должен четко информировать пользователя о допустимых границах. Текстовое поле, которое отвергает ввод пользователя после того, как он выполнил ввод, не может считаться ограничивающим элементом управления. Если пользователь должен выразить выбор числовым значением в определенных границах, предоставьте ему элемент управления, сообщающий об этих границах и предотвращающий ввод недопустимых значений. Такую возможность дает ползунок. Ползунок позволяет пользователю определять числовые значения в относительных терминах, а не в результате непосредственного ввода с клавиатуры. Но для ввода точных значений лучше подходят

счетчики.

Счетчики

Счетчик состоит из небольшого поля ввода и двух прикрепленных к нему кнопок (рис. 12). Благодаря счетчикам грань между ограничивающими и неограничивающими элементами ввода данных становится размытой. Маленькие кнопки со стрелками позволяют пользователю изменять значение в поле редактирования небольшими шагами. Эти шаги могут выполняться до определенного предела: значение не может превысить максимум, установленный программой, или стать меньше установленного минимума. Если пользователь пожелает ввести определенное число, он может сделать это за счет прямого ввода числа в поле редактирования.

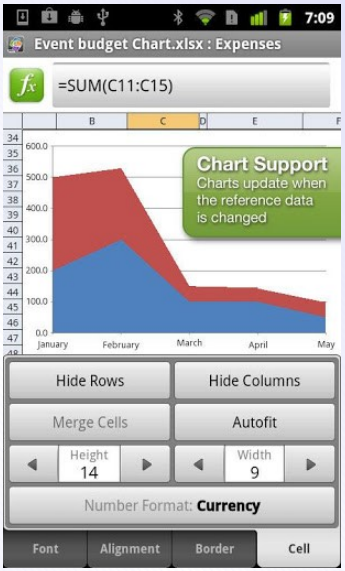


Рисунок 12. Реализация счетчиков в Android-приложении Quickoffice

Рукоятки и ползунки

Рукоятки и ползунки очень эффективно расходуют экранное пространство, и оба этих элемента управления замечательно справляются с задачей обеспечения визуальной обратной связи по настройкам (рис. 13). Ползунки и рукоятки применяются в основном в качестве ограничивающих элементов управления ввода. Например, ползунки - превосходное средство для действий, связанных с масштабированием.

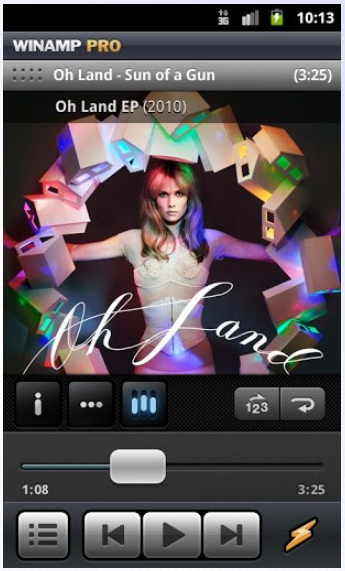


Рисунок 13. Ползунок в Android-приложении Winamp осуществляет перемотку воспроизводимой композиции

Неограничивающие элементы ввода

Пожалуй, главный неограничивающий элемент ввода - поле ввода текста. Этот простейший элемент управления позволяет пользователям набирать любые алфавитно-цифровые строки. Как правило, поля ввода - это небольшие области, внутри которых можно набрать одно-два слова, но они могут быть реализованы и в виде довольно сложных текстовых редакторов.

Когда пользователю предложено неограничивающее текстовое поле ввода, которое при этом принимает лишь строки определенного формата, вероятно, имеется необходимость помочь пользователям вводить "допустимые"строки. Имеется множество стандартных форматов вводимых данных - даты, телефонные номера, почтовые индексы, номера социального страхования. Ключ к успешному проектированию элемента ввода с проверкой данных - в хорошо развитой обратной связи с пользователем.

Элементы управления отображением

Элементы управления отображением используются для управления визуальным представлением информации на экране. Типичными примерами элементов отображения являются разделители и полосы прокрутки. Сюда же входят разделители страниц, линейки, направляющие, сетки и рамки.

Текстовые элементы

Вероятно, самый простой элемент управления отображением - элемент вывода текстовой информации, который отображает текстовое сообщение в некоторой позиции на экране. Он предоставляет текстовые метки для других элементов управления и выводит данные, которые не могут или не должны быть изменены пользователем. Единственная серьезная проблема этого элемента состоит в том, что он зачастую исполь-

зуется там, где должны присутствовать элементы ввода (и наоборот).

Полосы прокрутки

Полосы прокрутки служат важной цели - они позволяют осмысленным образом помещать большие объемы информации внутри рамок окон и панелей. К сожалению, они расходуют экранное пространство и ими сложно манипулировать. Однако замечательное преимущество полосы прокрутки состоит в создании контекста текущего положения в окне. Бегунок полосы прокрутки указывает текущее положение и нередко масштаб "территории доступной для прокрутки.

Разделители

Разделители - удобный инструмент для разделения главного окна приложения на несколько связанных между собой панелей, в каждой из которых можно просматривать, изменять или переносить ту или иную информацию. Подвижные разделители всегда должны сообщать о своей подвижности посредством изменения формы курсора. Однако следует проявлять осторожность, выбирая, какие именно разделители должны стать подвижными. В общем случае разделитель не должен перемещаться таким образом, чтобы содержимое панели становилось непригодным к использованию.

Выдвижные панели

Выдвижные панели - это панели приложения, которые можно открывать и закрывать в одно действие. Выдвижные панели - замечательное место для элементов управления и функций, которые используются совместно с основной рабочей областью приложения, но не столь часто. Выдвижные панели более удобны, чем диалоговые окна, так как не закрывают основное окно (рис. 14).

Рисунок 14. Скриншоты популярной игры "Cut the Rope". Новостная панель появляется, если потянуть за кольцо

13.4 Рекомендации по проектированию GUI под Android

Рекомендации разработчиков. Android Guideline

Когда платформа Android только появилась, не было никаких рекомендаций по разработке дизайна, поэтому все разработчики проектировали внешний вид приложений по своему вкусу. Отсутствие единого стиля сказалось на интерфейсах не лучшим образом, многие программы были откровенно некрасивы и неудобны. Кроме того, операционная система Android работает на устройствах с различными экранами, и разработчику необходимо помнить, что его приложение должно масштабироваться под различные параметры смартфонов и планшетов.

В настоящее время существует стандарт Android Design, и, если вы хотите, чтобы ваше приложение стало по-настоящему популярным и нужным, настоятельно рекомендуем его придерживаться. Далее мы рассмотрим основные принципы дизайна. Разумеется, в рамках этого курса невозможно учесть все нюансы. В списке источников есть ссылка на рекомендации от Android User Experience Team, к сожалению, все на английском языке.

Приведем выдержки из рекомендаций по дизайну:

- Реальные объекты гораздо веселее, чем кнопки и меню. Позвольте людям манипулировать знакомыми вещами! Тогда работа будет эффективнее.
- Картинки работают быстрее, чем слова.

- Используйте короткие фразы, состоящие из простых слов. Люди часто пропускают предложения, если они слишком длинные.
- Никогда не теряйте пользовательскую информацию. Если человеку придется вводить данные повторно, велика вероятность того, что он откажется использовать ваше приложение.
- Если объекты похожи, они должны выполнять сходные действия.
- Показывайте только то, что необходимо пользователю именно в этот момент.
- Выводите пользователю сообщения, только если вопрос действительно важен.
- Делайте важные вещи быстро.
- Разбивайте сложные задачи на несколько простых шагов.
- Будьте вежливы и корректны в общении с пользователем.
- Пользователь всегда должен быть уверен в том, что он знает, где сейчас находится. На любом шаге он должен иметь возможность вернуться назад, даже если это прервет выполнение какой-то задачи.
- Используйте интерфейсные элементы, которые будут работать в любой ситуации.
- Самый главный принцип - НЕ УСЛОЖНЯЙТЕ пользователю жизнь!

Обзор интерфейса

Приведем выдержки из рекомендаций по дизайну приложений для Android. Сделаем краткий обзор интерфейса операционной системы.

Домашний экран - это настраиваемая пользователем область, которая может содержать иконки приложений, папки и *виджеты*. Смартфон может иметь несколько домашних экранов, навигация между ними осуществляется с помощью перелистывания влево или вправо.

На домашнем экране в нижней в центре нижней части есть кнопка для открытия экрана приложений. Экран приложений позволяет пользователю запустить любую из установленных программ. Если устройство было использовано для отладки в процессе разработки, то приложение тоже окажется в этом списке и его можно будет вызвать даже после отключения от компьютера. Если приложение было использовано недавно, его можно найти в списке недавно использованных приложений, который вызывается нажатием на третью кнопку на панели внизу (см. рис. 15).

Рисунок 15. Домашний экран, экран всех приложений и список недавно использовавшихся приложений

В нижней и верхней частях экрана находятся системные панели, предназначенные для размещения уведомлений и навигации по устройству. Нижняя панель (Navigation Bar) предназначена для навигации на тех устройствах, которые не имеют аппаратных навигационных клавиш (все современные устройства). Верхняя часть экрана (Status Bar) предназначена для вывода различных сведений, например, времени, уровня заряда батареи, сигнала сотовой сети, а так же информационных сообщений.

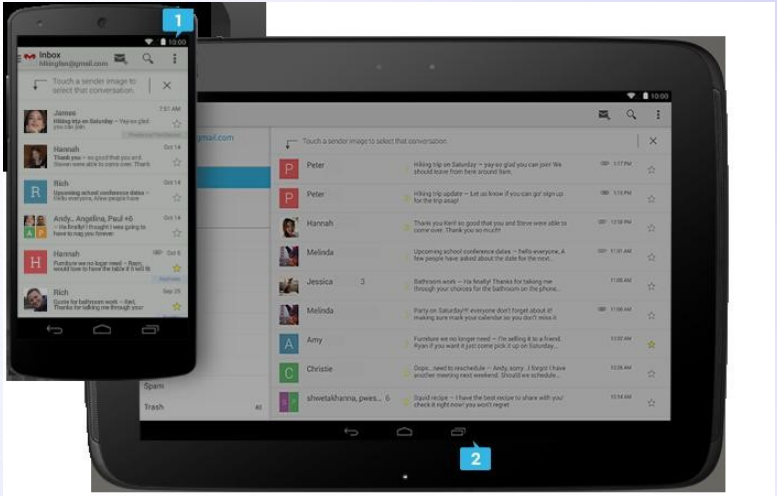


Рисунок 16. Информационная (1) и навигационная (2) панели

Уведомления - это быстрые сообщения, к которым пользователь может получить доступ в любое время из информационной панели. Это могут быть сообщения об обновлениях или других важных сообщениях, которые не настолько серьезны, чтобы прерывать работу пользователя. Но к ним легко можно получить доступ, потянув вниз верхнюю панель. Нажатие на уведомление вызывает соответствующее сообщение.

Шрифты

В дизайне Android используются традиционные типографические инструменты, такие как масштаб, разреженность и выравнивание по сет-

ке. Успешное применение этих выразительных средств помогает пользователю воспринимать информацию быстрее. В версии Android 4.0 Ice Cream Sandwich была представлена шрифтовая гарнитура без засечек Roboto, специально разработанная для экранов с высоким разрешением. Набор шрифтов доступен для бесплатной загрузки. Гарнитура включает в себя прямое и наклонное начертания для шрифтов различной ширины (см. рис. 17).

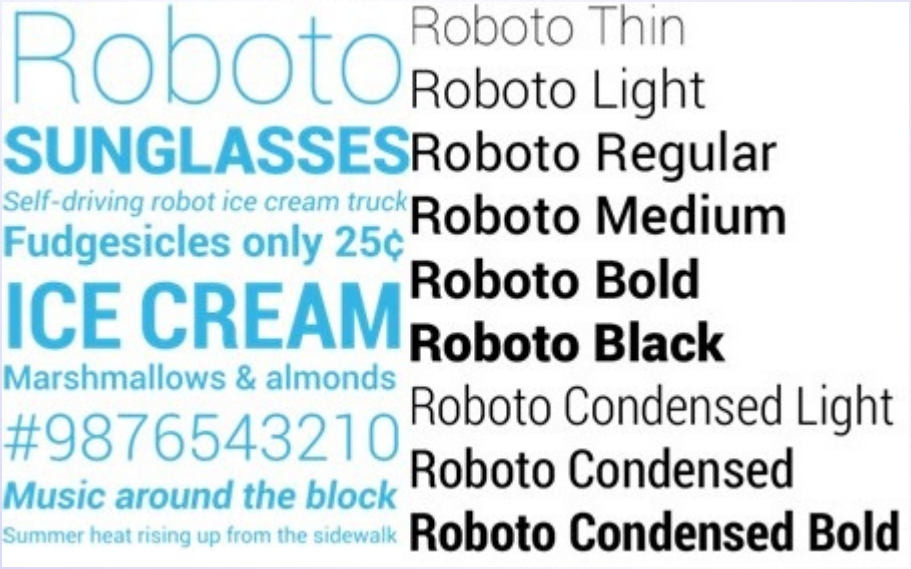


Рисунок 17. Шрифт Roboto и его возможные варианты

Масштабирование

Устройства различаются не только физическими размерами. Важным параметром является плотность экрана (DPI - количество точек на дюйм). Выделяют несколько категорий плотности экрана для Android-устройств: LDPI, MDPI, HDPI, XHDPI, XXHDPI, и XXXHDPI. Чтобы элементы интерфейса имели одинаковый физический размер на экранах разных устройств, компания Google ввела абстрактную единицу измерения - DP (независимый от разрешения пиксель). Один DP равен одному пикселю на экране типа MDPI. Устройства, имеющие меньше 600dp по короткой стороне, считаются телефонами, в противном случае мы говорим о планшетах (см. рис. 18).

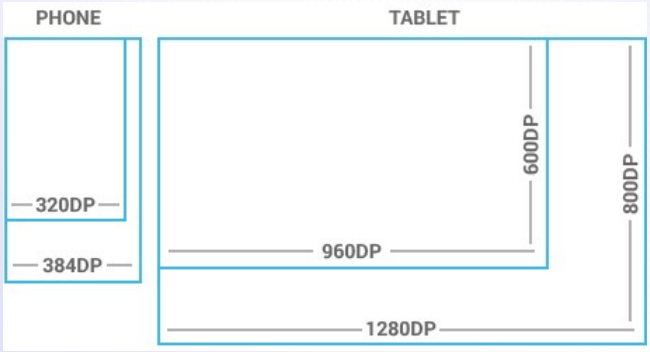


Рисунок 18. Размеры экранов телефонов и планшетов

Соответствие размеров экранов и их плотностей представлено в таблице 2.9:

Таблица 2.9. Плотности и размеры экранов

№	**Обозначение**	**Название**	**Соответствие**	**1 dp =**
1	LDPI	Low density	120 dpi	0,75 пикселя
2	MDPI	Medium density	160 dpi	1 пиксель
3	HDPI	High density	240 dpi	1,5 пикселя
4	XHDPI	Extra-high density	320 dpi	2 пикселя
5	XXHDPI	Extra-extra!-high density	480 dpi	3 пикселя
6	XXXHDPI	Extra-extra-extra!-high density	640 dpi	4 пикселя

Минимальный размер элемента управления - 48dp. Такое значение обусловлено тем, что на реальном устройстве оно соответствует 7-10 миллиметрам. При управлении кончиками пальцев такой размер является минимальным для отделения нужного элемента от всех остальных. Если какой-то из размеров элемента управления должен быть больше, чем 48dp, рекомендуется делать его размеры кратным этому значению (см. рис. 19).

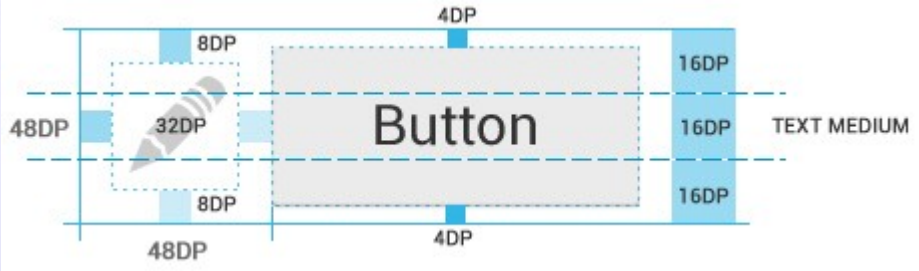


Рисунок 19. Размеры элемента управления кратны 48dp

Расстояние между элементами управления рекомендуется делать кратным 8dp (см. рис. 20).

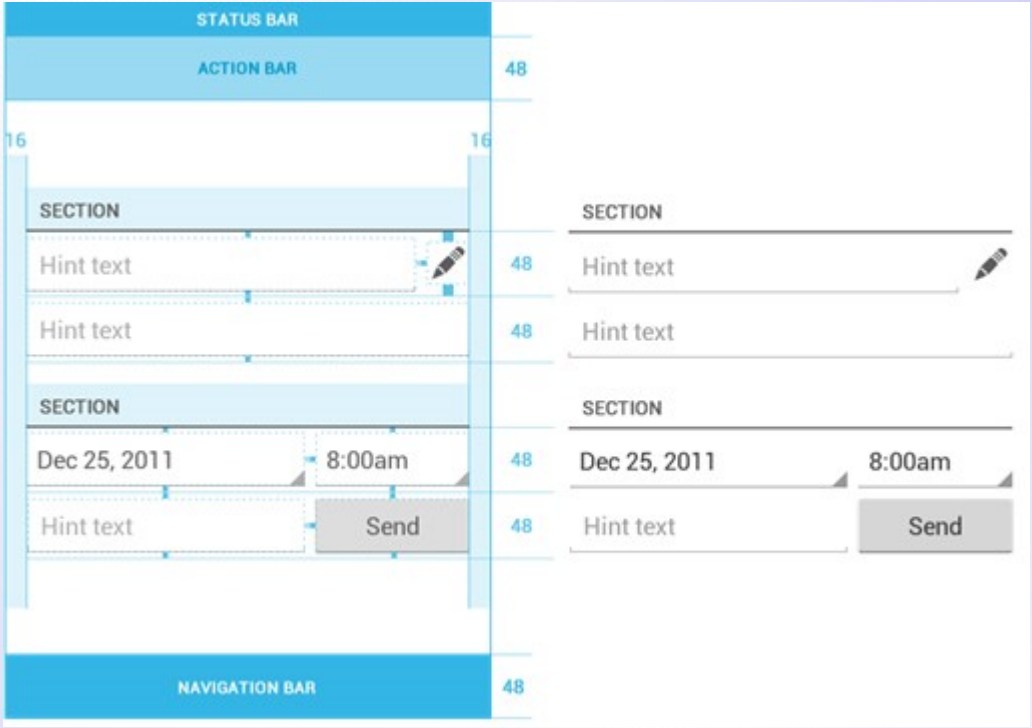


Рисунок 20. Пример расположения элементов управления

Прочие нюансы дизайна для Android рассмотрены в лабораторной работе.

§14. Основы разработки многооконных приложений

Аннотация: В прошлых лекциях мы рассмотрели особенности разработки приложений для ОС Android и настройки их интерфейсов. Однако все рассмотренные примеры вписывались в рамки экрана отдельно взятого устройства. Что делать в случаях, когда это условие не может быть соблюдено? В лекции рассказывается о работе с диалоговыми окнами, уведомлениями и всплывающими подсказками. Приведены особенности разработки приложений, содержащих несколько *активностей*, а так же способы перемещения между ними в запущенном приложении. Лекция может быть использована как часть курса, так и отдельно от него в целях углубления знаний по разработке многооконных Android-приложений.

Скриншоты приложений взяты из магазина приложений Google Play, официального сайта для разработчиков Android или сделаны самостоятельно с использованием смартфона Мегафон SP-A20i Mint на платформе Intel Medfield.

Презентацию к данной лекции можно скачать здесь.

14.1 Многооконные приложения

Для мобильных приложений главным ограничением является размер экрана устройства. Очень часто невозможно разместить все элементы

полнофункционального приложения так, чтобы их можно было увидеть одновременно. Очевидным решением этой проблемы является разделение интерфейса на части по какому-либо принципу. Основные пути решения этой проблемы:

- Использовать различные сообщения (диалоговые окна, уведомления, всплывающие подсказки). Этот способ наиболее прост и не требует редактирования файла *манифеста*, однако очевидно, что так можно решить только часть задач.

- Использовать в одном приложении несколько *активностей*. Способ универсальный и подходит для любых приложений, однако прежде чем его реализовывать, необходимо очень хорошо продумать структуру будущего приложения. Здесь требуется редактировать манифест и организовать переключение между различными активностями удобным для пользователя способом.

- Разместить компоненты на активности таким образом, что в нужный момент можно будет легко переключиться на работу с другой частью интерфейса.

Каждый способ имеет свои нюансы использования. Рассмотрим их более подробно.

14.2 Работа с диалоговыми окнами

Диалоговые окна

Диалог - это небольшое окно, позволяющее пользователю получить или ввести дополнительную информацию. Диалоговое окно занимает только часть экрана и обычно используется в модальном режиме. Это означает, что работа приложения приостанавливается до момента, пока пользователь не закроет диалоговое окно. При этом ему, возможно, потребуется ввести какие-то данные или просто выбрать один из вариантов ответа (см. рис. 1).

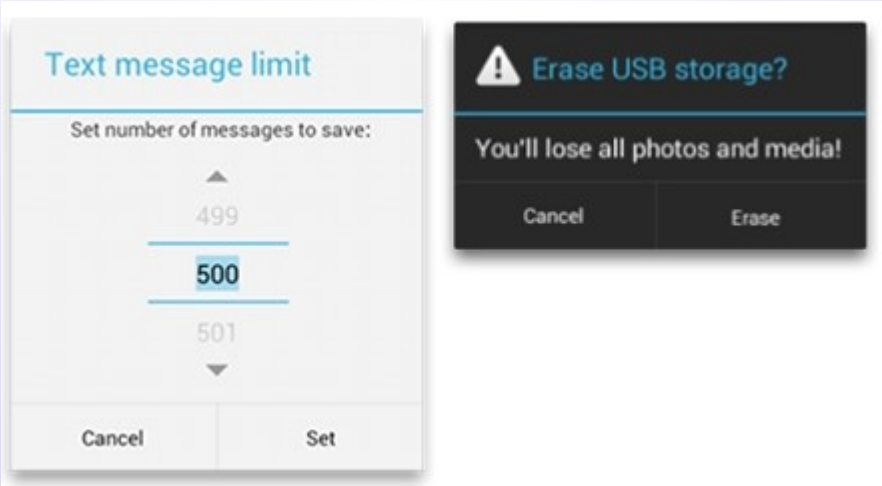


Рисунок 1. Примеры диалоговых окон

В ОС Android можно выделить три вида *диалоговых окон*:

- **Класс Dialog и его производные**. Помимо традиционного набора диалоговых окон, он содержит несколько дополнительных вариантов, в которых используются возможности сенсорного интерфейса (см. рис. 1 слева). Диалоги этого типа не создают новых *активностей* и их не нужно регистрировать в файле *манифеста* (см. следующие разделы лекции), что существенно упрощает разработку. Однако они работают в модальном режиме и требуют немедленного ответа пользователя, поэтому для простого информирования рекомендуется использовать сообщения следующих двух типов.
- **Уведомления (notifications)**. Это сообщения, которые отображаются в верхней панели в области уведомлений. Для того чтобы прочитать это сообщение, необходимо на домашнем экране потянуть вниз верхнюю шторку. Пользователь может это сделать в любой момент времени, следовательно, уведомления стоит использовать, когда сообщение является важным, однако не требует немедленного прочтения и ответа.
- **Всплывающие подсказки (toasts)**. Сообщения, которые появляются прямо на экране приложения, перекрывая его интерфейс, и через некоторое время (обычно несколько секунд) автоматически пропадают. Их рекомендуется использовать для простых уведомлений, не требующих ответа пользователя, но важных для продолже-

ния его работы.

Использование класса Dialog

Класс Dialog является базовым для *диалогов* и редко используется напрямую. Рекомендуется применять производные от этого класса:

- AlertDialog. Диалоговое окно может содержать заголовок, до трех кнопок, список выбираемых значений или настраиваемое содержимое. Пример на рис. 1 справа.
- DatePickerDialog или TimePickerDialog. Диалоговое окно с предопределенным интерфейсом, позволяющее выбрать дату или время.
- ProgressDialog. Показывает диалоговое окно, содержащее линейку процесса выполнения какого-то действия. В рекомендациях по дизайну для Android советуют использовать вместо него компонент ProgressBar.

Существует возможность создавать собственные диалоговые окна с использованием класса DialogFragment в качестве контейнера. В таком случае можно контролировать его поведение. Обратите внимание, что минимальной версией, поддерживающей DialogFragment напрямую, является Android 3.0 (API level 11). Если вы хотите использовать возможности этого класса на более ранних версиях, необходимо добавить библиотеку Support Library в ваше приложение.

Рассмотрим создание *диалогового окона* на примере класса AlertDialog. Существует множество вариантов диалоговых окон этого класса, однако все они содержат следующие три части (см. рис. 2):

1. **Заголовок**. Не является обязательным элементом и должен быть использован, только если содержательная часть занята детализированным сообщением, списком или чем-то еще. Если нужно сделать небольшое сообщение или вопрос, не стоит снабжать его выделенным заголовком.
2. **Содержательная часть**. Здесь может быть сообщение, список или какой-то другой настраиваемый компонент.
3. **Управляющие кнопки**. Диалог может содержать не больше трех кнопок. Если элементы содержательной части являются кликабельными, можно вообще обойтись без кнопок (см. рис. 3).

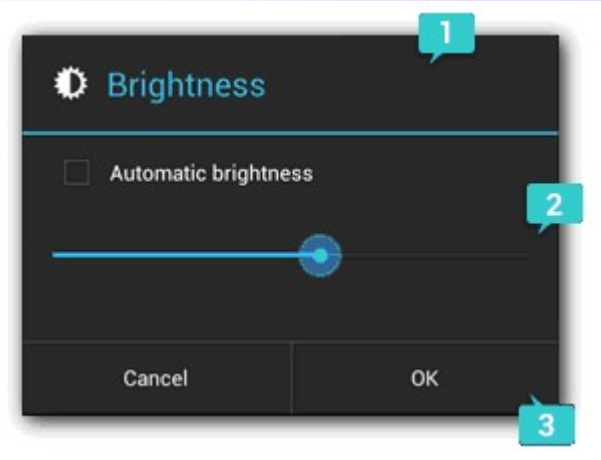


Рисунок 2. Компоновка диалогового окна (класс AlertDialog)

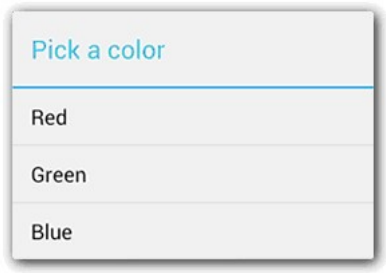


Рисунок 3. Можно выбрать один из трех перечисленных цветом непосредственно нажатием на его название

Возможности использования *диалоговых окон* будут рассмотрены в лабораторной работе более подробно.

Уведомления

Уведомления являются неотъемлемой частью дизайна Android- приложений. На рис. 4 показаны уведомления в свернутом и развернутом виде.

Рисунок 4. Уведомления. Слева - информационная панель со свернутыми уведомлениями, справа эти же уведомления развернуты

Существует два варианта отображения уведомлений в развернутом виде - нормальный и расширенный (доступен начиная с Android 4.1).

Состав уведомления в нормальном виде представлен на рис. 5 Высота уведомления составляет 64 dp. Уведомление содержит следующие части:

1. Заголовок.
2. Большая иконка.
3. Текст сообщения.
4. Информация о сообщении.
5. Маленькая иконка приложения.
6. Время (или дата), когда было отправлено сообщение.

Рисунок 5. Стандартное уведомление

Уведомление появляется в расширенном виде, только если оно находится вверху списка уведомлений, или же когда пользователь сделал жест с целью его увеличения. Расширенное уведомление (см. рис. 6) включает в себя те же пункты, что и обычное, но при этом дополнительно содержит детализированную область (на рисунке отмечено номером 7). Это может быть, например, картинка до 256 dp высотой, блок текстовой информации или что-то еще.

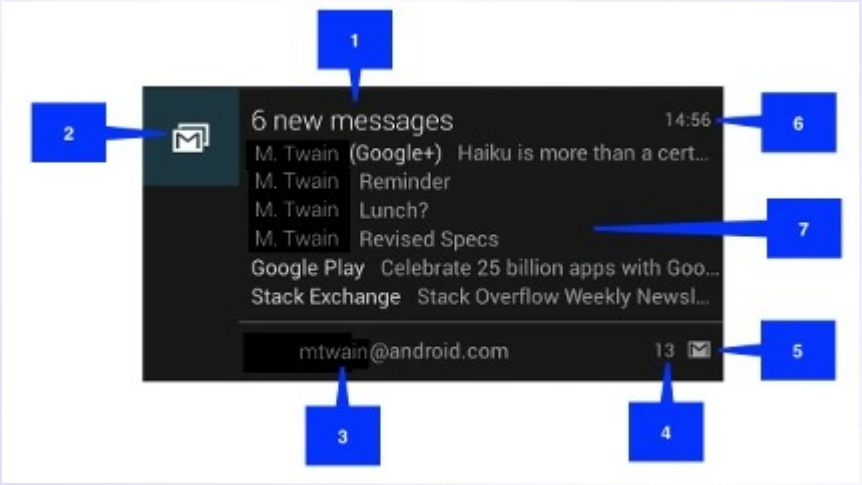


Рисунок 6. Расширенное уведомление

Всплывающие подсказки

Всплывающие подсказки помогают отобразить обратную связь с действиями пользователя. Они занимают минимум места на экране и быстро исчезают (см. рис. 7). Поэтому рекомендуется использовать их для простого уведомления пользователя, когда не требуется получить от него ответа. Всплывающие подсказки могут появляться в любом месте экрана, что позволяет делать их работу более эффективной.

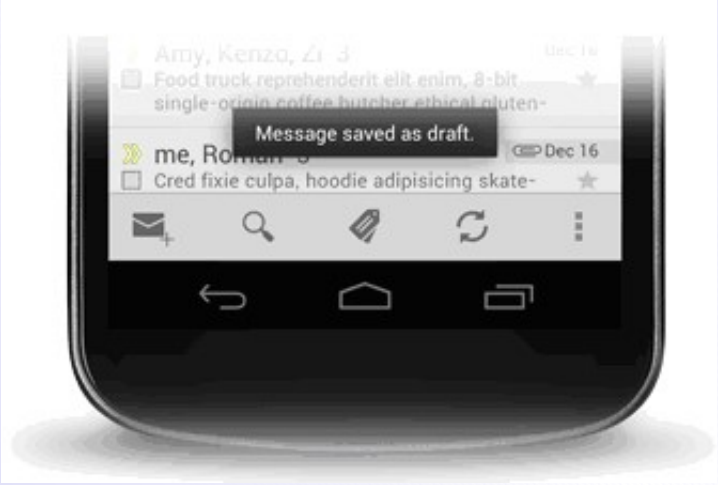


Рисунок 7. Всплывающие подсказки

14.3 Особенности разработки приложения, содержащего несколько активностей

Приложения, содержащие несколько *активностей*, используются в самых разных сферах. При проектировании такого приложения следует уделить большое внимание распределению его функционала по разным активностям. С одной стороны, не стоит перегружать экран информацией, а с другой - нужна ли *активность*, содержащая только одно *поле* для ввода? Может быть, стоит ее заменить диалоговым окном?

Существует два основных способа переключения между активностями:

- **При помощи кнопок и других элементов управления**. Не требует перестройки мышления у программистов, которые имеют большой опыт разработки десктопных приложений, а так же у пользователей, привыкших к действиям в стиле "нажал на кнопку, получил результат". Однако этот способ не является наиболее подходящим для сенсорных экранов и требует от опытного пользователя смартфона совершения лишних движений.
- **С использованием сенсорного экрана смартфона**. Основная идея состоит в том, что весь экран мобильного устройства можно использовать в качестве управляющего элемента, и, нажимая на отдельные его участки, пользователь может инициировать те или иные действия. Более подробно возможности жестового интерфейса

будут рассмотрены далее, в этой и следующей темах.

Какой из способов выбрать, зависит от конкретной задачи.

Существует ряд правил расположения интерфейсных элементов в зависимости от их важности. Так, кнопку, выполняющую важное действие (например, отправку письма), не стоит располагать в том месте, где она может быть случайно нажата. В то же время *управляющие* элементы, используемые наиболее часто, должны быть расположены наиболее удобным для нажатия образом. Скорее всего, перемещение между *активностями* будет использоваться не очень часто, поэтому рекомендуется располагать кнопки, *управляющие* этими действиями, в верхней части экрана. Одновременно с этим неплохо продублировать нажатия кнопок перелистыванием между активностями.

В любом случае для вызова другой активности необходимо вручную править *файл манифеста*. Для каждой новой активности необходимо занести информацию о ее имени и названии xml-файла, в котором она описана (см. *листинг 7.1*). Обратите внимание, что при загрузке приложения первой появляется *активность*, чье описание находится первым в манифесте! Если вы хотите изменить порядок загрузки активностей, необходимо поместить новую *активность* на первое *место*.

```
<?xml version="1.0"encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.myproject.screen"
```

```
android:versionCode="1"
android:versionName="1.0»

<uses-sdk
android:minSdkVersion="8"
android:targetSdkVersion="17"/>

<application
android:allowBackup="true"
android:icon="@drawable/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme»
<activity
android:name="com.myproject.screen.MainActivity"
android:label="@string/app_name»
<intent-filter>
<action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
<activity
android:name="com.myproject.screen.AboutActivity"
android:label="@string/about_title»
```

```
</activity>
<activity
android:name="com.myproject.screen.SecondActivity"
android:label="@string/title_activity_second»
</activity>
</application>
</manifest>
```

Листинг 7.1. Исправленный файл манифеста.

14.4 Перелистывание (Swipe)

Существует способ разместить на *активности* больше элементов, чем одновременно помещается на экран, иными словами, отображать по очереди несколько экранов, используя только одну *активность*. В этом случае не нужно править *файл манифеста* - *активность* только одна. Однако для каждого экрана необходимо сделать свой xml-*файл* с его описанием.

Такой способ размещения элементов удобен и программисту, и пользователю. Разработчик может организовать перемещение между частями активности и с помощью кнопок, и с помощью перелистывания. В качестве примера подобного интерфейса можно привести *приложение Twitter* (см. рис. 8).

Рисунок 8. Главный экран приложения Twitter. Можно переключаться между экранами, используя жест горизонтальной прокрутки или с помощью кнопок в верхней части

§15. Использование возможностей смартфона в приложениях

В этом курсе проделана уже немалая работа: установлена и настроена *среда разработки*, созданы первые приложения, хочется двигаться дальше. Разработка мобильных приложений под *Android* имеет ряд особенностей, часть из них мы уже рассмотрели, часть ожидают рассмотрения в ближайших темах. Данная тема, как видно из названия, посвящена возможностям смартфонов и их использованию в приложениях.

Особенностью большинства мобильных устройств является наличие сенсорного экрана и возможность управления пальцем (touch-interface), очевидно, что это необходимо учитывать и использовать при разработке приложений. *Смартфон*, если уж появляется у человека, становится его спутником всегда и везде, в связи с этим, довольно часто используется, как фотоаппарат или проигрыватель музыки, а также смартфоны все чаще становятся инструментами ориентирования на местности.

В данной теме предполагается рассмотрение вопросов разработки приложений, ориентированных на тач-*интерфейс*, работу со звуком, использование камеры и глобальных систем позиционирования.

15.1 Отличительные особенности смартфонов

Пришло время поговорить о наиболее интересных возможностях смартфонов, которые можно использовать в приложениях. Ни для кого не секрет, что *смартфон* является "умным телефоном": предполагает обязательное наличие операционной системы и возможность установки дополнительных приложений, существенно расширяющих функционал устройства. С одной стороны, *смартфон* выполняет все привычные функции мобильного телефона и, благодаря компактным размерам, всегда под рукой. С другой стороны, благодаря наличию процессора и операционной системы, позволяет выполнять многие функции полноценного компьютера. Дополнительно ко всему, смартфоны обладают рядом интересных особенностей, не характерных для телефонов и компьютеров.

Для начала обратим внимание на экран смартфона. В современных смартфонах экран занимает практически всю *площадь* передней панели устройства, имеет высокое разрешение и является чувствительным к прикосновениям. Благодаря такой чувствительности, для взаимодействия с устройством и его приложениями можно использовать виртуальные *элементы управления*, чаще всего кнопки, отображаемые на экране. В связи с чем отпадает необходимость в физических кнопках. В смартфонах реализуется, так называемый, touch-*интерфейс* - интерфейс, основанный на виртуальных элементах управления, выбор которых выполняется простым касанием, а также на использовании жестов (gestures).

Если точек касания несколько (т. е. используется несколько пальцев), такой *интерфейс*, уже называется multi-touch.

Еще одна особенность смартфонов состоит в том, что для большинства их владельцев не последнюю роль играет возможность использования этого "умного телефона"в качестве аудио или видеоплеера, поэтому современные устройства становятся все более и более мультимедийными. В первой лекции обсуждалось, что в состав платформы *Android* входит набор библиотек для обработки *мультимедиа Media* Framework, в котором реализована *поддержка* большинства общих медиа-форматов. В связи с чем, в приложения, разрабатываемые для смартфонов под управлением *Android*, можно интегрировать *запись* и воспроизведение аудио и видео, а также работу с изображениями.

Важной и часто используемой особенностью смартфонов является наличие камеры, которая позволяет снимать все самое интересное: от первых шагов ребенка до падения метеорита. Телефон всегда под рукой и готов к работе, в связи с этим количество фотографий и небольших видеороликов резко увеличилось, и любое интересное событие в жизни индивидуума может быть запечатлено и сохранено для потомков. С ростом возможностей получения фото и видео материалов увеличивается потребность в приложениях, способных работать с этими материалами. Платформа *Android* позволяет разрабатывать такие приложения, которые предоставляют пользователям возможности делать фотоснимки или записывать видео, каким-то образом обрабатывать полученные ма-

териалы и использовать их далее.

Большинство смартфонов оснащены *GPS*-модулем, а некоторые даже комбинированным модулем *GPS*/ГЛОНАСС, что позволяет использовать такое устройство в качестве инструмента для ориентирования на местности. Во многих случаях *смартфон* с установленным соответствующим программным обеспечением вполне может заменить *GPS* навигатор. В разрабатываемых приложениях иногда бывает очень полезно добавить возможность получения координат устройства и хозяина, если оба находятся в одном месте, и использовать эти *координаты* для каких-либо целей. Например, уже существуют приложения, которые позволяют отслеживать параметры человека (спортсмена) во время преодоления некоторых расстояний бегом, на велосипеде, на лыжах и т. д. Такое *приложение* работает во время тренировки (устройство должно перемещаться вместе со спортсменом), по окончанию можно получить полную статистику маршрута: *точное время* в пути, *расстояние*, подъемы/спуски, среднюю скорость, потраченные калории и т. д. Заметим, что большая часть информации опирается на данные, полученные со спутников *GPS*.

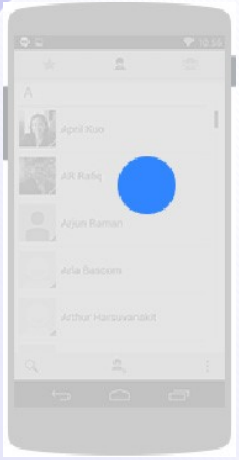
Рассмотрение особенностей смартфонов будет неполным, если оставить без внимания датчики и сенсоры, которыми оснащены большинство устройств. Эти микроустройства обеспечивают связь смартфона с окружающей средой и добавляют новые удивительные функции. С помощью датчика приближения, например, можно отключать подсветку

экрана при приближении телефона к уху пользователя во время разговора, блокировать экран, чтобы не было возможности случайно нажать на отбой. *Акселерометр* может использоваться для смены ориентации экрана, для управления в играх, особенно симуляторах, а также в качестве шагомера. Датчик освещенности позволяет регулировать яркость экрана. *Гироскоп* может применяться для определения более точного позиционирования устройства в пространстве.

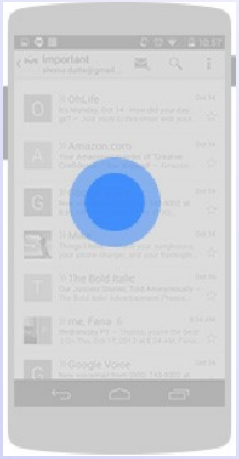
Все рассмотренные особенности в совокупности увеличивают привлекательность смартфонов, позволяют разработчикам создавать приложения с разнообразными, полезными, интересными и иногда неожиданными функциями. Далее в лекции рассмотрим перечисленные возможности смартфонов более подробно и узнаем как можно их использовать при разработке приложений.

15.2 Сенсорное (touch) управление

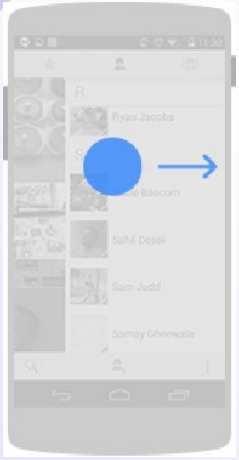
В этом разделе лекции рассмотрим возможности добавления сенсорного управления в мобильные приложения под *Android*. Сенсорное управление подразумевает использование сенсорных жестов для взаимодействия с приложением. Ниже представлен набор жестов, поддерживаемый системой *Android*.



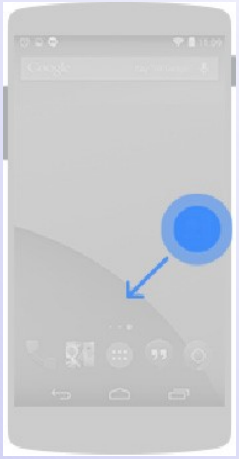
Касание (touch).
Использование: Запуск действия по умолчанию для выбранного элемента.
Выполнение: нажать, отпустить.



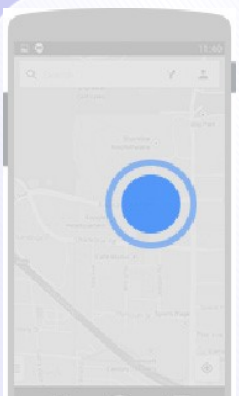
Длинное касание (long touch).
Использование: Выбор элемента. Не стоит использовать этот жест для вызова контекстного меню.
Выполнение: нажать, ждать, отпустить.



Скольжение или перетаскивание (swipe or drag).
Использование: Прокрутка содержимого или навигация между элементами интерфейса одного уровня иерархии.
Выполнение: нажать, переместить, отпустить.



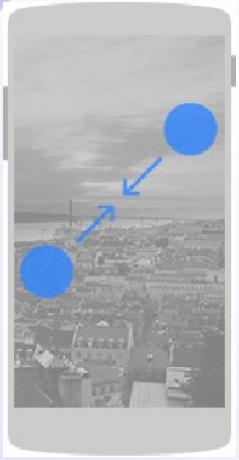
Скольжение после длинного касания (long press drag).
Использование: Перегруппировка данных или перемещение в контейнер.
Выполнение: длительное касание, переместить, отпустить.



Двойное касание (double touch).
Использование: Увеличение масштаба, выделение текста.
Выполнение: быстрая последовательность двух касаний.

Перетаскивание с двойным касанием (double touch drag).
Использование: Изменение размеров: расширение или сужение по отношению к центру жеста.
Выполнение: касание, следующее за двойным касанием со смещением вверх или вниз при этом:

- смещение вверх уменьшает размер содержимого;
- смещение вниз увеличивает размер содержимого.



Сведение пальцев (pinch close).
Использование: уменьшение содержимого, сворачивание.
Выполнение: касание экрана двумя пальцами, свести, отпустить.

Разведение пальцев (pinch open).
Использование: увеличение содержимого, разворачивание.
Выполнение: касание экрана двумя пальцами, развести, отпустить.

О возможности управлять приложением с помощью сенсорных жестов можно говорить в том случае, когда *приложение* способно распознать, что под набором касаний экрана скрывается некоторый жест и выполнить соответствующее действие. Процесс распознавания жеста обычно состоит из двух этапов: сбор данных и *распознавание* жеста. Рассмотрим эти этапы подробнее.

Сбор данных о сенсорных событиях

Основные действия, которые может произвести пользователь при взаимодействии с сенсорным экраном: коснуться экрана пальцем, переместить палец по экрану и отпустить. Эти действия распознаются системой Android, как сенсорные события (touch-события).

Каждый раз при появлении сенсорного события инициируется вызов метода onTouchEvent(). Обработка события станет возможной, если этот метод реализован в классе *активности* или некоторого компонента, иначе событие просто игнорируется.

Жест начинается, при первом касании экрана, продолжается пока система отслеживает положение пальцев пользователя и заканчивается получением финального события, состоящего в том, что ни один палец не касается экрана. Объект MotionEvent, передаваемый в метод onTouchEvent(), предоставляет детали каждого взаимодействия. Рассмотрим основные константы класса MotionEvent, определяющие сенсорные события:

- MotionEvent.ACTION_DOWN - касание экрана пальцем, является начальной точкой для любого сенсорного события или жеста;
- MotionEvent.ACTION_MOVE - перемещение пальца по экрану;
- MotionEvent.ACTION_UP - поднятие пальца от экрана.

Приложение может использовать предоставленные данные для распознавания жеста.

Можно реализовать свою собственную обработку событий для распознавания жеста, таким образом можно работать с произвольными жестами в приложении. Если же в приложении необходимо использовать стандартные жесты, описанные выше, можно воспользоваться классом GestureDetector. Этот класс позволяет распознать стандартные жесты без обработки отдельных сенсорных событий.

Распознавание жестов

Android предоставляет класс GestureDetector для распознавания стандартных жестов. Некоторые жесты, которые он поддерживает включают: onDown(), onLongPress(), onFling() и т. д. Можно использовать класс GestureDetector в связке с методом onTouchEvent(). Подробно распознавание поддерживаемых жестов рассмотрено в первой части лабораторной работы в этой теме.

Начиная с версии 1.6, Android предоставляет API для работы с жестами, который располагается в пакете android.gesture и позволяет сохра-

нять, загружать, создавать и распознавать жесты. Виртуальное устройство Android (AVD), начиная все с той же версии 1.6, содержит предустановленное приложение, которое называется Gesture Builder и позволяет создавать жесты. После создания жесты сохраняются на SD карте виртуального устройства и могут быть добавлены в приложение в виде бинарного ресурса.

Для распознавания жестов необходимо добавить компонент GestureOverlayView, в XML файл *активности*. Этот компонент может быть добавлен как обычный элемент графического интерфейса пользователя и встроен в компоновку, например RelativeLayout. С другой стороны он может быть использован, как прозрачный слой поверх других компонентов, в этом случае в XML файле активности он должен быть записан, как корневой элемент.

Кроме всего вышеперечисленного, для использования собственных жестов в приложении необходимо реализовать интерфейс OnGesturePerformedListener и его метод onGesturePerformed(). Подробно создание и использование собственных жестов рассмотрено во второй части лабораторной работы в этой теме.

15.3 Работа с мультимедиа

Мультимедиа библиотека *Android* включает поддержку воспроизведения МНОЖЕСТВА наиболее распространенных форматов, что позво-

ляет легко использовать в приложениях аудио, видео и изображения. Можно проигрывать аудио или видео из медиа файлов сохраненных как ресурсы приложения (*raw* ресурсы), из файлов, расположенных в файловой системе или из потока данных, получаемого через сетевое соединение, для всего этого используется MediaPlayer *API*.

Замечание: проигрывать аудиофайлы можно только на стандартном устройстве вывода, невозможно воспроизводить аудио во время звонка.

Актуальная *информация* о поддерживаемых форматах аудио и видео приводится по ссылке: здесь.

Для воспроизведения аудио и видео *Android* предоставляет *класс* MediaPlayer. Причем при работе с аудиоконтентом этот *класс* позволяет воспроизводить необработанные данные, т. е. возможно проигрывание динамически генерируемого аудио.

Диаграмма жизненного *цикла* экземпляра класса MediaPlayer представлена на рис. 1. Овалы представляют состояния объекта MediaPlayer, дуги показывают вызовы каких методов необходимо выполнить, чтобы сменить состояние объекта MediaPlayer . Дуги с одной стрелкой представляют вызовы синхронных методов, с двумя стрелками - вызовы асинхронных методов.

В ходе жизненного цикла объект MediaPlayer проходит через несколько состояний:

- **бездействие (Idle)** - создан экземпляр класса MediaPlayer для создания может использоваться оператор new или вызов метода reset() (см. рис. 1);

  (источник: здесь.)

- **инициализирован (Initialized)** - задан источник медиа- информации, для задания источника используется метод setDataSource();
- **ошибка (Error)** - появилась какая-то ошибка, например, не поддерживаемый аудио/видео формат, слишком высокое разрешение, чтобы вывести объект из этого состояния, необходимо вызвать метод reset();
- **подготовка (Preparing)** - MediaPlayer занимается подготовкой медиаисточника к воспроизведению, подготовка инициируется методом prepareAsync();

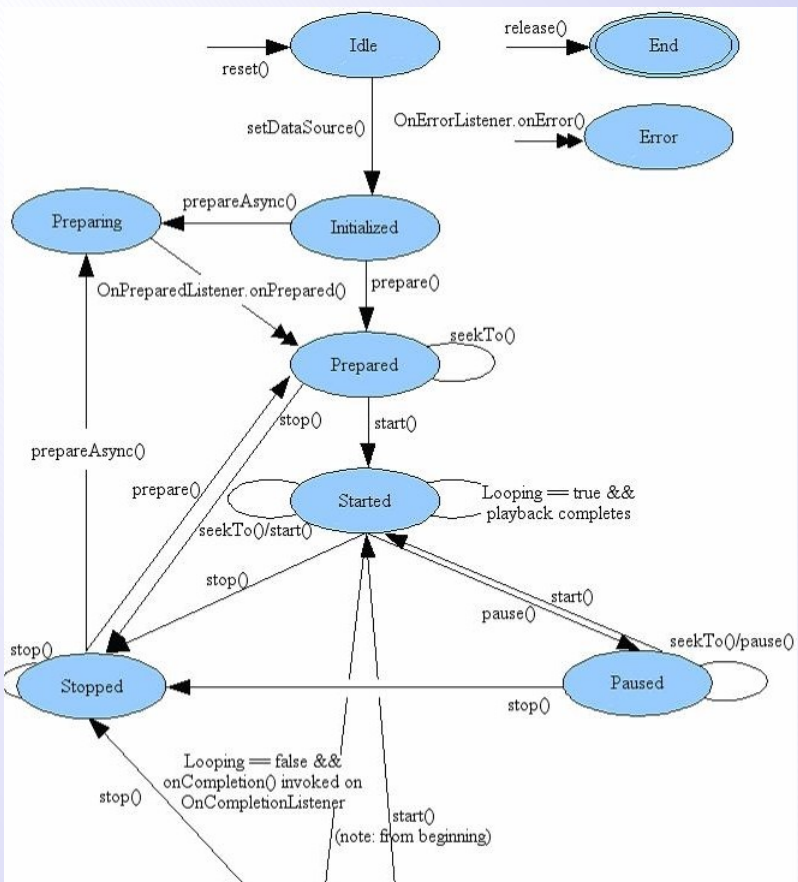


Рисунок 1. Жизненный цикл экземпляра класса MediaPlayer

- **готов (Prepared)** - состояние готовности к воспроизведению, может быть достигнуто двумя способами:
  - синхронный способ: вызов метода prepare(), который переводит объект в готовое состояние;
  - асинхронный способ: срабатывание метода onPrepared() интерефейса OnPreparedListener() в состоянии подготовки, как реакция на событие готовности;
- **запущен (Started)** - выполняется воспроизведение медиа-контента, в это состояние объект переходит после вызова метода start();
- **приостановлен (Paused)** - воспроизведение приостановлено, MediaPlayer переходит в это состояние после вызова метода pause();
- **остановлен (Stopped)** - воспроизведение остановлено, MediaPlayer переходит в это состояние после вызова метода stop();
- **воспроизведение завершено (Playback Completed)** - достигнут конец воспроизводимого содержания, в это состояние объект переходит после срабатывания метода onCompleted() интерфейса-слушателя OnCompitionListener, как реакции на конец воспроизводимого материала;

  Замечание: из состояний Paused, Playback Completed можно вернуться к воспроизведению вызовом метода start(). Из состояния

Stopped прежде, чем вернуться в состояние воспроизведения, необходимо пройти через подготовку медиа-содержимого.

Вызов метода seekTo() позволяет поменять место воспроизведения.

- **конец (End)** - конец жизненного цикла MediaPlayer объекта, в это состояние объект переходит после вызова метода release().

Для получения более детальной информации см. ссылки: 1 ; 2.

Для записи аудио и видео *Android* предоставляет класс MediaRecorder. *Диаграмма* жизненного *цикла* экземпляра класса MediaRecorder представлена на рис. 2. Овалы представляют состояния объекта MediaPlayer, дуги показывают вызовы каких методов необходимо выполнить, чтобы сменить состояние объекта MediaPlayer. Дуги с одной стрелкой представляют вызовы синхронных методов, с двумя стрелками - вызовы асинхронных методов.

(источник: здесь.)

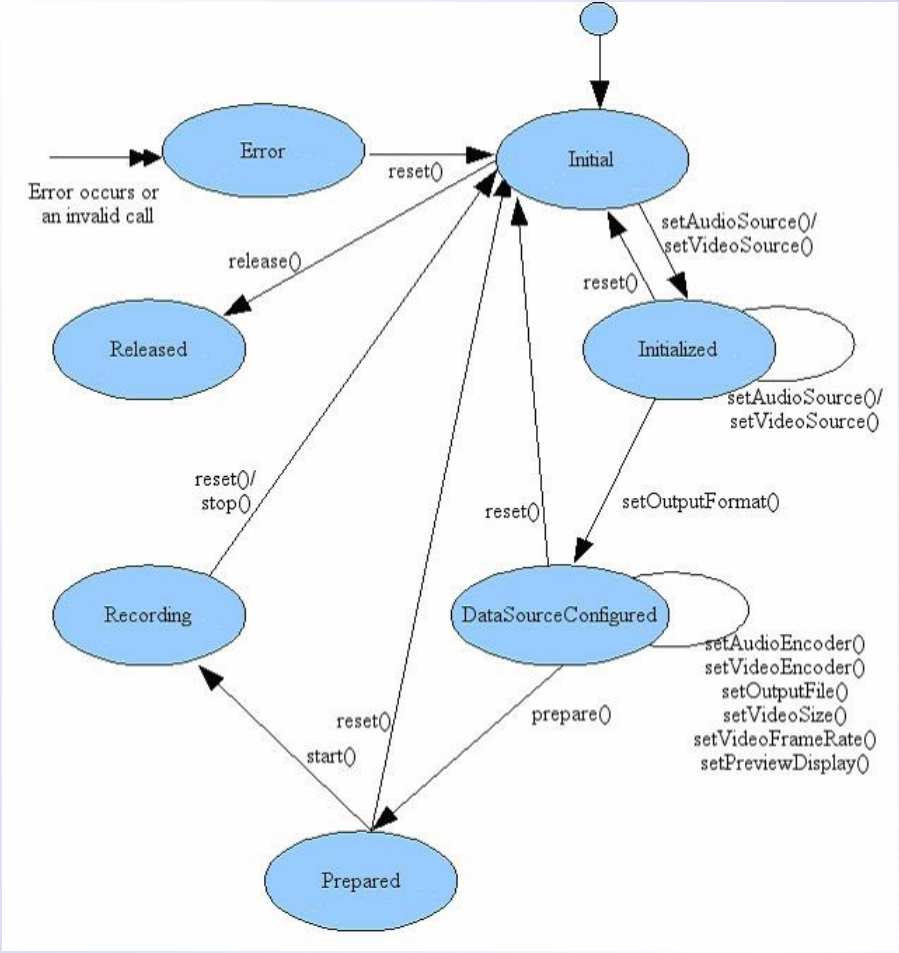


Рисунок 2. Жизненный цикл экземпляра класса MediaRecorder

В ходе жизненного *цикла объект* MediaRecorder проходит через несколько состояний:

начальное (Initial)	- создан объект класса MediaRecover, для создания может использоваться оператор new или вызов метода reset() (см. рис. 2);
инициализирован (Initialized)	- объект MediaRecover готов к использованию, в данное состояние объект переходит после вызова одного из методов setAudioSource() или setVideoSource(), которые задают источники аудио или видео для записи;
сконфигурирован приемник данных для записи (Data Source Configured)	- задаются основные свойства приемника данных, состояние инициируется методом setOutputFormat(), для настройки свойств должны быть выполнены некоторые методы из списка: setAudioEncoder(), setVideoEncoder(), setOutputFile(), setVideoSize(), setVideoFrameRate(), setPreviewDisplay();
готов (Prepared)	- состояние готовности к записи, инициируется методом prepare();
записывает (Recording)	- идет запись, инициируется вызовом метода start();

освобожден (Released) - запись завершена, все ресурсы освобождены.

Для получения более детальной информации см. ссылки: 1; 2.

15.4 Использование встроенной камеры

Платформа *Android* включает поддержку камеры, доступной на устройстве, позволяющей приложениям получать фотографии и записывать видео. Для решения этих задач, существует два способа:

1. непосредственное обращение к камере;
2. использование намерений (Intent) для вызова существующего приложения.

Рассмотрим основные относящиеся к делу классы:

Camera - класс, реализующий управление камерами устройства. Этот класс используется для получения фотографий или записи видео при создании приложения, работающего с камерой.

SurfaceView - класс, используемый для предоставления пользователю возможности предварительного просмотра.

MediaRecorder - класс, используемый для записи видео с камеры.

Intent - класс, содержащий абстрактное описание выполняемой операции, которое передается системе Android, а ОС сама находит и запускает необходимое приложение и возвращает результат его работы. Для работы с камерой используются два типа намерений:

- MediaStore.ACTION_IMAGE_CAPTURE - для запроса на выполнение фотоснимков;
- MediaStore.ACTION_VIDEO_CAPTURE - для запроса на запись видео.

Подробно процесс разработки приложения, позволяющего производить фото и видеосъемку рассмотрен в третьей части лабораторной работы к данной теме.

15.5 Взаимодействие с системами позиционирования

Системы позиционирования позволяют определить местоположение в некоторой системе координат, обычно определяются широта и долгота.

Так как *смартфон* является мобильным телефоном, ему доступны

методы, обычно используемые мобильными телефонами для определения своего местоположения.

- Во-первых, смартфон постоянно связывается с сотовой вышкой, в зоне действия которой он находится. Каждая сотовая вышка в мире имеет уникальный идентификатор, называемый идентификатором соты (Cell ID), а также для нее точно известны широта и долгота ее расположения. В связи с этим, смартфон, зная идентификатор соты, в которой он находится, может получить географические координаты центра этой соты. Радиусы сот варьируются в зависимости от того, насколько активный сетевой трафик ожидается в конкретном районе. Разумеется, такой способ позиционирования дает очень приблизительные результаты, что называется: "плюс-минус трамвайная остановка".

- Во-вторых, чаще всего смартфон оказывается в зоне действия более, чем одной сотовой вышки. В современных мобильных технологиях, начиная с поколения 2G, сотовая вышка может определить, с какого направления приходит сигнал. В случае, когда телефон находится в зоне действия двух или трех сотовых вышек, они могут выполнять триангуляцию его местоположения. Телефон может запросить у сети информацию о том, где он находится. Такая техника определения местоположения может быть очень точной и не требует установки дополнительного оборудования.

Дополнительно к возможностям определения местоположения, доступным обычным мобильным телефонам, большинство смартфонов укомплектованы спутниковыми системами глобального позиционирования (*Global Positioning System, GPS*). В настоящее время наиболее распространенными в мире системами глобального спутникового позиционирования являются: GPS, разработанная и реализованная в США, и система ГЛОНАСС (Глобальная навигационная спутниковая система), советская, а позже российская спутниковая система навигации. Многие смартфоны могут использовать сигналы сразу от двух навигационных систем, что позволяет серьезно увеличить надежность и точность определения координат, прежде всего, в городских условиях.

В *дополнение* к вышеперечисленным методам позиционирования, добавляется возможность использования сигналов WiFi, Bluetooth и *NFC*, а также внутреннего сенсора для более точной геолокации, особенно внутри помещений.

В этом разделе нас, в первую *очередь*, будет интересовать возможность добавления в приложения способностей определять *координаты* устройства и работать с картами. При создании приложений, учитывающих текущее местоположение, под *Android* можно воспользоваться *GPS* и определением местоположения в сети (с помощью *Network Location Provider*). Несмотря на то, что *GPS* дает более точные результаты, он не очень хорошо работает в помещениях (чаще не работает), он сильно расходует заряд батареи и скорость определения координат не всегда

соответствует ожиданиям пользователя. *Network Location Provider* определяет *координаты*, используя сигналы сотовых вышек и WiFi, может работать как на улице, так и внутри помещений, более экономно расходует заряд батареи и работает быстрее по сравнению с *GPS*. Для получения координат в приложении можно использовать оба способа или один из них на выбор.

Android предоставляет приложениям *доступ* к геолокационным возможностям мобильного устройства, через классы пакета *android.location*. Центральным классом этого пакета является *класс* LocationManager, который предоставляет *доступ* к системным *сервисам* для определения координат устройства.

В приложения можно добавлять карты, используя Google Maps *Android API*, которое автоматически управляет доступом к серверам Google Maps, загрузкой данных, отображением карт и сенсорными жестами на карте. Также можно использовать вызовы *API* для добавления маркеров, многоугольников и внешних прозрачных слоев, а также для изменения пользовательского представления отдельных участков карты.

Ключевым классом в Google Maps *Android API* является *класс* MapView, который отображает карту с данными полученными из сервиса Google Maps. Когда MapView имеет фокус, он может перехватывать нажатия клавиш и сенсорные жесты для выполнения автоматического перемещения и изменения масштаба карты, а также может управлять сетевыми запросами для получения дополнительных фрагментов кар-

ты. Этот *класс* так же предоставляет все элементы пользовательского интерфейса, необходимые для управления картой.

Google Maps *Android API* не является частью платформы *Android*, но доступен на любом устройстве с Google *Play Store*, работающем, начиная с *Android 2.2*, через Google *Play services*. Чтобы обеспечить возможность интеграции Google Maps в приложения, в *Android SDK* необходимо установить библиотеку Google *Play services*.

Подробнее вопросы добавления в приложения геолокационных возможностей и использование карт (Google Maps) рассмотрены в четвертой части лабораторной работы к данной теме.

15.6 Другие сенсоры и датчики

Большинство устройств, работающих под управлением *Android*, укомплектованы встроенными сенсорами, которые предоставляют исходные данные высокой точности. Сенсоры могут быть полезны в том случае, если необходимо регистрировать положение и перемещения, повороты устройства в трехмерном пространстве, а также изменения параметров окружающей среды.

Платформа *Android* поддерживает три категории сенсоров:

Датчики движения	Эти сенсоры измеряют силы ускорения и вращательные силы по трем осям. Эта категория включает акселерометры, гироскопы, датчики вектора вращения и сенсоры силы тяжести.
Датчики окружающей среды	Эти сенсоры измеряют различные параметры окружающей среды, такие как температура воздуха и давление, освещенность и влажность. Эта категория включает барометры, термометры и датчики освещенности.
Датчики положения	Эти сенсоры измеряют физическое положение устройства. Эта категория включает магнитометры и датчики ориентации устройства в пространстве.

Сенсоры могут быть реализованы аппаратно или программно. Аппаратно-реализованные датчики являются физическими элементами встроенными в *мобильное устройство*, они получают данные путем прямых измерений некоторых свойств, таких как ускорение, сила геомагнитного поля или изменение углов. Программно-реализованные датчики получают свои данные с одного или нескольких физических датчиков и вычисляют *значение*, которое от них ожидается.

Какие типы датчиков поддерживаются *Android* можно узнать по ссыл-

ке: здесь.

Android предоставляет набор классов и интерфейсов для работы с сенсорами. Эти классы и интерфейсы являются частью пакета *android.hardware* и позволяют выполнять следующие задачи:

- определять какие сенсоры доступны на устройстве;
- определять индивидуальные возможности сенсоров, такие как максимальное значение, производитель, требования к потребляемой энергии и разрешения;
- собирать данные с сенсоров и определять минимальную частоту, с которой выполняется сбор данных;
- подключать и отключать слушателей событий от датчиков, события состоят в изменении значений датчиков.

Для работы с датчиками *Android* предоставляет следующие классы и интерфейсы:

SensorManager -Этот класс может использоваться для создания экземпляра сервиса, связанного с сенсором. Также он предоставляет различные методы для доступа и составления списка сенсоров, подключения и отключения слушателей событий от сенсоров, сбора информации. Этот класс содержит константы, которые используются для задания точности сенсора, частоты получения данных и настройки датчиков.

Sensor -Этот класс используется для создания экземпляра датчика, предоставляет методы, позволяющие определить свойства сенсора.

SensorEvent -Система использует этот класс для создания объекта, соответствующего событию датчика и предоставляющего следующую информацию: данные сенсора; тип сенсора, который породил событие, точность данных и время появления события.

SensorEventListener -Данный интерфейс может использоваться для реализации двух методов, получающих уведомления (события датчиков), когда меняется значение сенсора или когда меняется точность сенсора.

Использование в приложении полученных от сенсоров данных будет рассмотрено в лабораторной работе темы 7. Подробнее об использовании сенсоров можно узнать по ссылке: здесь.

§16. Использование библиотек

Аннотация: Прежде чем браться за решение какой-то вспомогательной задачи, следует сначала выяснить, не была ли она решена кем-то ранее. Повторное использование кода позволяет сберечь ресурсы на выполнение проекта. Такие возможности предоставляют подключаемые библиотеки, рассмотрению возможностей которых посвящена данная тема. В лекции приведена классификация библиотек по их назначению и возможности их подключения. Рассматриваются некоторые популярные подключаемые библиотеки, как официальные, так и альтернативные. Затрагиваются вопросы безопасности использования библиотек. Лекция может быть использована как в рамках изучения данного курса, так и отдельно от него, если читатель желает подробнее ознакомиться с возможностью работы с подключаемыми библиотеками.

Скриншоты приложений взяты из магазина приложений Google Play или сделаны самостоятельно, в том числе с использованием смартфона Мегафон SP-A20i Mint на платформе Intel Medfield. Некоторые иллюстрации взяты с официальных сайтов.

16.1 Библиотеки

Использование библиотек

Библиотека (от англ. library) в программировании - сборник под-

программ или объектов, используемых для разработки программного обеспечения (ПО). Для ОС Android существует большое количество подключаемых библиотек. Их можно классифицировать в зависимости от их предназначения. Выделим следующие группы:

- **Библиотеки совместимости.** Они позволяют использовать возможности, появившиеся в какой-то версии ОС Android, на более ранних версиях платформы. Дело в том, что новые версии API выходят гораздо быстрее, чем в широком использовании оказываются устройства, поддерживающие эту версию. Разработчик с одной стороны должен ориентироваться на новые возможности и уметь их использовать, а с другой - стараться сделать так, чтобы приложение работало на максимальном количестве устройств. Библиотеки совместимости позволяют сделать это противоречие менее жестким.
- **Библиотеки специального назначения.** Используются для разработки игр, работы с социальными сетями, сбора статистики и в других случаях.
- **Библиотеки, предоставляющие дополнительные возможности.** В эту категорию можно отнести большое количество самых разных библиотек. Сюда можно отнести библиотеки рисования графиков, работы с изображениями, модифицированные элементы управления и многое другое.

Подключение библиотек

Библиотеки могут поставляться как в собранном и уже готовом к использованию виде (jar-файлы), так и в исходниках. Подключить библиотеку (файл *.jar) очень просто. Достаточно создать папку **libs** в проекте (на том же уровне, что и папки **src** и **res**) и копировать туда файл библиотеки (можно просто перетащить). Дальше необходимо добавить ее в проект через меню **Project -> Properties**.

Если библиотека представлена в виде исходного кода, необходимо ее предварительно собрать. Необходимо щелкнуть правой кнопкой по корневой папке проекта **- > Export: -> Java -> Runnable JAR file - >Указать класс для запуска -> Указать место сборки -> Finish** (см. рис. 1).

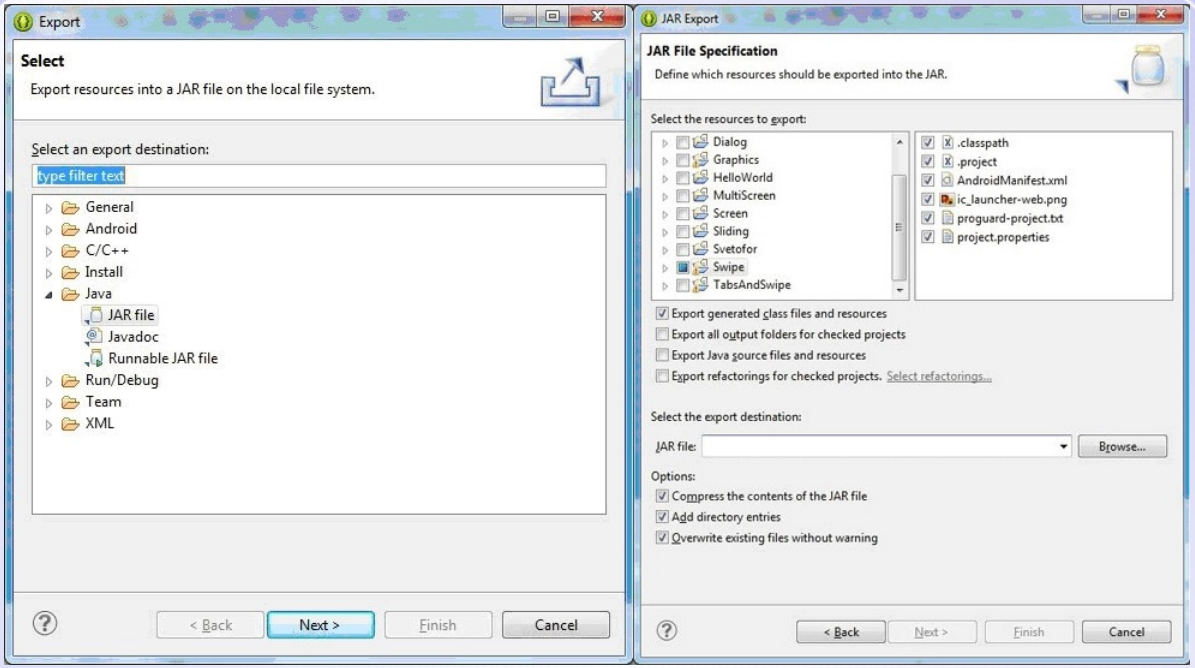


Рисунок 1. Сборка библиотеки из исходного кода

16.2 Обзор популярных библиотек

Android Support Library

Android Support Library - это набор *библиотек*, которые обеспечивают обратную совместимость новых API на более старых версиях платформы. Каждая библиотека из этого набора обладает обратной совместимостью к конкретному уровню Android API. Это означает, что ва-

ши приложения смогут использовать возможности библиотеки и быть запущены на устройствах Android 1.6 (API level 4) и выше.

Подключение библиотек поддержки в Android является хорошим тоном в разработке приложений, зависящих от версии и возможностей платформы. Использование возможностей Support Library поможет вам распространить ваше приложение для большего числа пользователей. Если вы используете примеры Android-приложений, вы можете заметить, что все они содержат по умолчанию одну или несколько библиотек поддержки.

О возможностях различных версий *Android Support Library* можно узнать на официальном сайте. Скачать и установить эти библиотеки можно с помощью Android *SDK Manager*, выбрав в разделе Extras нужные пункты (см. рис. 2).

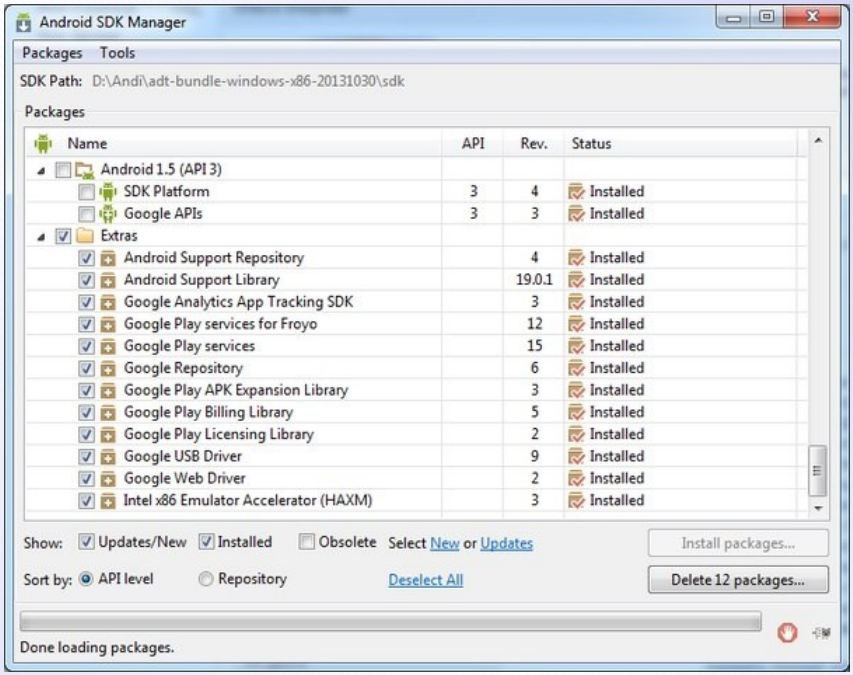


Рисунок 2. Подключение Android Support Library

При настройке обратной совместимости необходимо отредактировать файл манифеста, указав в нем минимальную версию Android SDK, которая необходима для запуска приложения, и основную (целевую) версию:

```
<uses-sdk
android:minSdkVersion="7"
android:targetSdkVersion="17"/>
```

Сторонние библиотеки

Помимо официальных и поддержанных Google библиотек совместимости, существуют аналогичные решения от сторонних разработчиков. Вы можете их использовать на свой страх и риск, но в некоторых случаях они предпочтительнее, так как обладают некоторыми дополнительными возможностями.

NineOldAndroids - один из примеров таких библиотек. Она предназначена для использования анимации, которая стала доступна только с версии Honeycomb (Android 3.0), на всех более ранних платформах. Она поддерживает различные возможности анимации и очень удобна в использовании (см. рис. 3). Главным преимуществом этой библиотеки является то, что она работает для всех версий Android, начиная с 1.0.

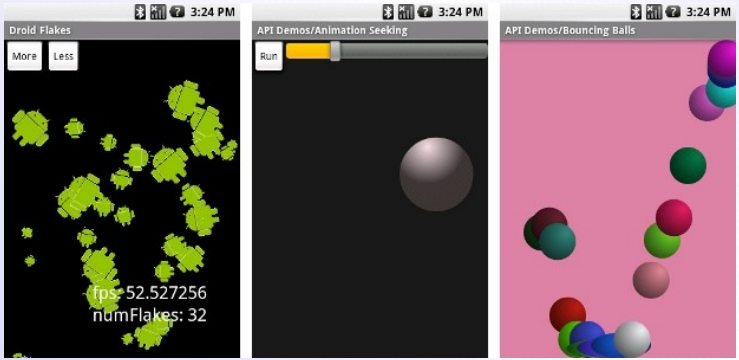


Рисунок 3. Использование библиотеки NineOldAndroids

Другим примером подобного решения является автономная библиотека **ActionBarSherlock**. С ее помощью можно использовать нативный компонент ActionBar, появившийся только в версии Android 4.0, в более ранних (2.x и выше). Ее можно загрузить с официального сайта. Там же содержатся подробные указания по работе с этой библиотекой, имеются примеры. На рис. 4 представлена работа приложения, использующего библиотеку ActionBarSherlock, на устройствах со старыми версиями.

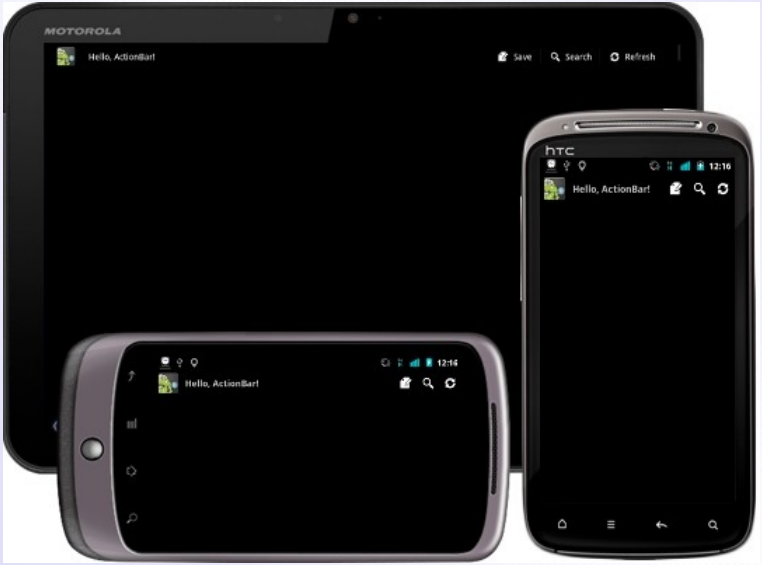


Рисунок 4. Использование библиотеки ActionBarSherlock

Библиотеки специального назначения

Yandex.Metrica for Apps - набор библиотек для сбора статистики использования мобильного приложения. Метрика показывает актуальную статистику об использовании приложения. *сервис* позволяет отвечать на вопросы об аудитории и выделять любые ее сегменты. Инструменты помогают понять, как люди пользуются приложением. SDK позволяет отслеживать следующие данные:

- информация об устройстве;
- информация о сессиях;
- информация об источнике перехода пользователя на страницу скачивания приложения;
- действия, выполненные пользователем в приложении;
- местоположение пользователя;
- ошибки, возникающие во время использования приложения;
- собственные события;
- другие данные (например, количество пользователей, установивших приложение).

Подробные инструкции по добавлению в приложение Яндекс.Метрики и работе с ней есть на официальном сайте.

Facebook SDK for Android - официальная библиотека Facebook для Android. Позволяет писать сообщения на стену, читать и менять статусы, смотреть ленту друзей и многое другое. Официальный сайт содержит большое количество примеров и указаний по разработке приложений (см. рис. 5).

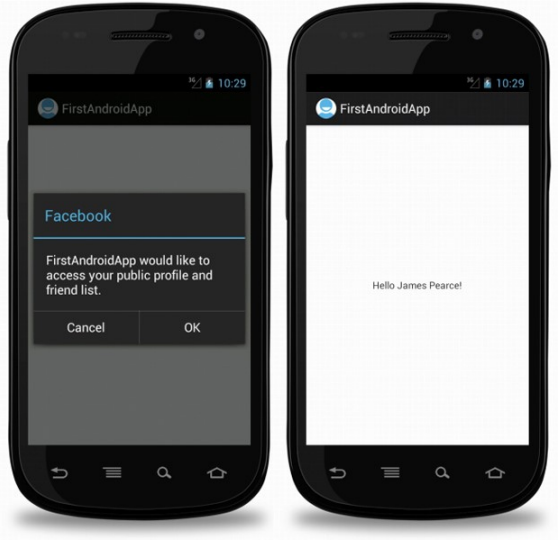


Рисунок 5. Пример приложения, подготовленного с использованием Facebook SDK for Android

Прикладные библиотеки

К этой категории можно отнести различные библиотеки, предоставляющие дополнительные возможности.

Universal Image Loader for Android - мощная и гибкая библиотека, предназначенная для загрузки, кеширования и отображения картинок в Android. Подробности на сайте.

Возможности:

- Многопоточная загрузка изображений.
- Широкие возможности настройки и конфигурирования.
- Кеширование загруженных изображений как в оперативной памяти, так и на карте.
- Поддержка виджетов.
- Поддерживает Android 2.0 и выше.

Рисунок 6. Пример работы с библиотекой Universal Image Loader for Android

jsoup: Java HTML Parser предназначена для парсинга HTML-страниц. Предоставляет очень удобный API для извлечения данных и манипуляции с ними, используя DOM, CSS и методы в стиле jQuery. Поддерживает спецификации HTML5 и позволяет парсить страницы так же, как это делают современные браузеры.

Возможности:

- Может принимать в качестве параметра URL, файл или строку.
- Находит и извлекает данные, используя DOM и селекторы CSS.
- Позволяет манипулировать HTML-элементами, атрибутами и текстом.

- Выводит чистый HTML.

Примеры ее использования есть на официальном сайте.

Android Holo ColorPicker - удобная библиотека, позволяющая выбирать цвет с использованием цветового колеса, выполненная в официально рекомендованном стиле Holo. Сайт библиотеки содержит описание работы с ней и необходимые ссылки.

Рисунок 7. Пример использования Android Holo ColorPicker

Библиотека **MapNavigator** предназначена для работы с картами Google Maps. Позволяет определять направления и отображать маршрут на карте. Работает только с Google Maps v2. Скачать можно на официальном сайте.

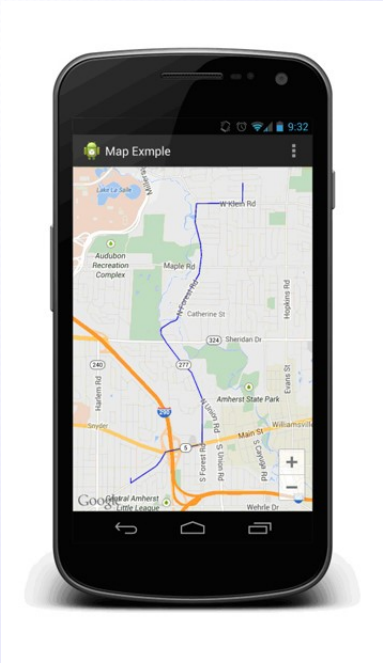


Рисунок 8. Пример использования библиотеки MapNavigator

AChartEngine - библиотека, предназначенная для построения графиков. Позволяет строить графики различных типов:

- Линии графиков функций.
- Поточечные графики.
- Гистограммы.
- Круговые диаграммы.
- Пузырьковые диаграммы.
- Комбинированные диаграммы.
- Другие виды диаграмм и графиков.

Все типы диаграмм поддерживают несколько рядов данных.

Сайт разработчика содержит подробную документацию, оформленную в стиле Javadoc pages, примеры использования библиотеки, а также ее исходный код.

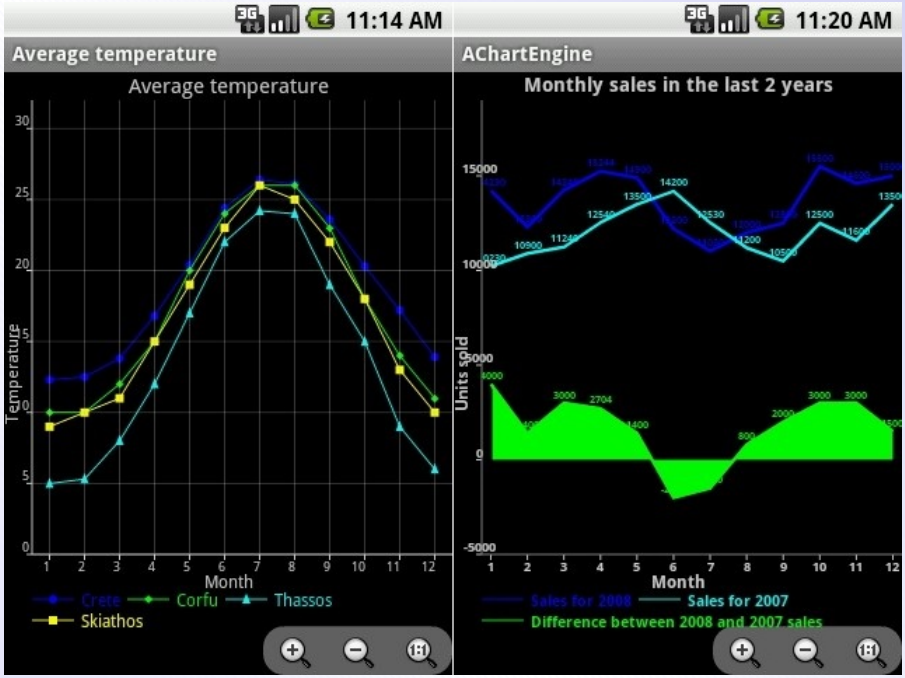


Рисунок 9. Пример использования библиотеки AChartEngine

Разумеется, мы рассмотрели лишь малую долю существующих библиотек. Обзор не претендует на полноту, но это не так важно, так как его целью было представить многообразие возможностей, открывающихся перед разработчиком. Кроме того, большое количество разнообразных библиотек описаны на сайте.

16.3 Безопасность использования подключаемых библиотек

Подключаемые *библиотеки* являются очень удобным инструментом, облегчающим труд программиста. Однако разработчики приложений, использующие сторонние библиотеки подобного рода, часто не подозревают об их проблемах с безопасностью. Библиотека может содержать возможности, которые могут использоваться злоумышленниками в преступных целях.

Например, в октябре 2013 года была опубликована статья с результатами исследования, согласно которому популярная у разработчиков библиотека, предоставляющая возможность отображения рекламы в приложениях, может использоваться для сбора информации и запуска вредоносного кода. Исследователи не раскрыли истинного названия библиотеки, зато описали возможный вред. Например, она может запускать на устройстве произвольный код, извлекать текстовые сообщения, *список* контактов и вызовов, передавать секретную информацию пользователя в виде простого текста по протоколу *HTTP*, использовать камеру без ведома пользователя, запускать вредоносные java-скрипты. *Злоумышленник* может превратить эту библиотеку в ботнет, перехватывая ее трафик и отправляя вредоносные команды и код [37].

При выборе библиотеки следует соблюдать осторожность. Если библиотека поставляется в виде исходников и о ней мало информации, не

лишним будет просмотреть ее код в поисках странных функций, которые не нужны для ее функционирования. Но для уже собранной библиотеки *анализ* исходников может стать непростой задачей. Поэтому при выборе библиотеки следует соблюдать ряд правил:

- Не использовать скомпрометированные библиотеки. Если о какой-то библиотеке появляются сведения, что она может содержать вредоносный код, следует отказаться от ее использования в новых проектах и по возможности пересмотреть ее применение в уже существующих.
- С осторожностью использовать библиотеки из сомнительных источников.
- Обязательно ознакомиться с форумами и сайтами, где могут обсуждаться библиотеки. Кроме того, это может помочь вам подобрать наиболее подходящее решение для вашей конкретной задачи.
- По возможности просмотреть исходники.
- Применять другие правила информационной безопасности, которые могут иметь значение в каждом конкретном случае.

§17. База данных и мультимедия в Android

Современное *программирование* трудно представить без использования баз данных, рано или поздно в процессе развития приложения появляется осознание необходимости долговременного хранения и обработки структурированной информации. Данная лекция посвящена рассмотрению вопросов, связанных с использованием баз данных SQLite в приложениях, разрабатываемых под *Android*. *Базы данных* SQLite являются основой построения рабочей и функциональной программы, в которой необходимо работать с большими объемами структурированной информации.

Далее в лекции перейдем к рассмотрению таких интересных тем, как создание графических изображений и анимации, а также работа с этими элементами. Платформа *Android* предоставляет разнообразные способы для добавления в приложения и использования графики и анимации.

Очень часто мобильные устройства помогают "скоротать время"в очередях, в ожидании транспорта и многих других ситуациях, часто возникающих в современной жизни. Проще всего в такие моменты занять себя несложной игрой, в связи с этим тема разработки игр для мобильных устройств стала довольно популярна в последнее время. Разумеется, разработка игр дело серьезное, но даже отдельному разработчику *по* силам создать игру, способную увлечь пользователя. В данной теме рассмотрим основные принципы создания игр для смартфонов, в лабораторной

работе рассмотрим процесс создания несложной игры.

17.1 Основы работы с базами данных, SQLite

SQLite - небольшая и при этом мощная система управления базами данных. Эта система создана в 2000 году, ее разработчик доктор Ричард Хипп (Dr. Richard Hipp). В настоящее время является одной из самых распространенных *SQL*-систем управления базами данных в мире. Можно выделить несколько причин такой популярности SQLite: она бесплатная; она маленькая, примерно 150 Кбайт; не требует установки и администрирования. Подробнее см. .

***База данных* SQLite** - это обычный *файл*, его можно перемещать и копировать на другую систему (например, с телефона на рабочий *компьютер*) и она будет отлично работать. *Android* хранит *файл базы данных* приложения в папке (см. рис. 1):

data/data/packagename/databases/,

где packagename - *имя пакета*, в котором расположено *приложение*.

Для доступа к этому файлу необходимо запускать команды *SQL*, *Android* с помощью вспомогательных классов и удобных методов скрывает часть деталей, но все-таки необходимо иметь хотя бы минимальные знания об *SQL*, чтобы пользоваться этими инструментами.

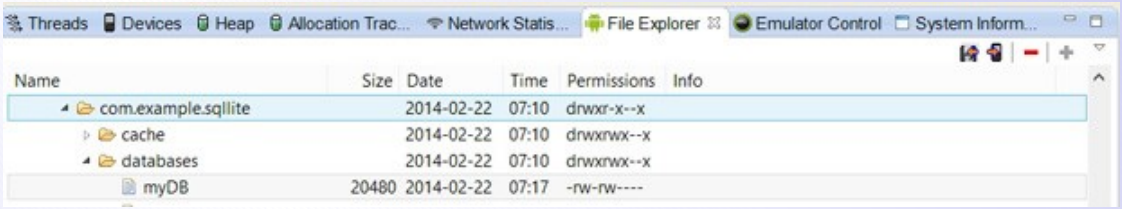


Рисунок 1. Расположение файла базы данных SQLite

Обращения к базе данных *SQL* выполняются посредством запросов, существует три основных вида *SQL* запросов: *DDL, Modification* и Query.

- **DDL запросы**. Такие запросы используются для создания таблиц. Каждая таблица характеризуется именем и описанием столбцов, которое содержит имя столбца и тип данных. В файле базы данных может быть несколько таблиц.

  Пример запроса для создания таблицы:

  ```
  create Table_Name (
  _id integer primary key autoincrement,
  field_name_1 text,
  field_name_2 text);
  ```

  Первый столбец обозначен, как primary key (первичный ключ), т.е. уникальное число, которое однозначно идентифицирует строку. Слово autoincrement указывает, что база данных будет автоматически увеличивать значение ключа при добавлении каждой записи, что и

обеспечивает его уникальность. Существует договоренность первый столбец всегда называть _id, это не жесткое требование SQLite, однако может понадобиться при использовании *контент-провайдера* в Android.

Стоит иметь в виду, что в SQLite, в отличие от многих других баз данных, типы данных столбцов являются лишь подсказкой, т. е. не вызовет никаких нареканий попытка записать строку в столбец, предназначенный для хранения целых чисел или наоборот. Этот факт можно рассматривать, как особенность базы данных, а не как ошибку, на это обращают внимание авторы SQLite.

- **Modification запросы**. Такие запросы используются для добавления, изменения или удаления записей.

  Пример запроса на добавление строки:

  insert into Table_Name values(null, value1, value2);

  В этом случае значения разместятся в соответствующие столбцы таблицы, первое значение задается для поля _id и равно null, т. к. SQLite вычисляет значение этого поля самостоятельно.

  При добавлении можно указывать столбцы, в которые будут размещаться значения, остальные столбцы заполнятся значениями по умолчанию, в этом случае можно добавлять элементы в измененном порядке. Пример такого запроса:

insert into Table_Name(field_name_2, field_name_1)
values(value2, value1);

В этом случае добавляются значения только в поля field_name_1 и field_name_2, причем изменен порядок следования полей, а вместе с этим и порядок следования значений, иногда это бывает удобно.

Примеры запросов на изменение строки:

update Table_Name set Field_Name_1 = value;

поменяет значение столбца Field_Name_1 на value во всей таблице;

update Table_Name set Field_Name_1 = value where _id = smth;

поменяет значение столбца Field_Name_1 только в той строке, _id которой равен smth.

Примеры запросов на удаление строк:

delete from Table_Name;
delete from Table_Name where Field_Name_1 = smth;

первый запрос удаляет таблицу целиком, второй - только те строки, в которых столбец Field_Name_1 имеет значение smth.

- **Query запросы**. Такие запросы позволяют получать выборки из таблицы по различным критериям. Пример запроса:

  select from Table_Name where (_id = smth);
  select Field_Name_1,

Field_Name_2 from Table_Name
Field_Name_1 = smth);

Первый *запрос* выводит строку с _id равным smth, второй - выводит два элемента Field_Name_1 и Field_Name_2 строк, в которых Field_Name_1 равен smth.

Вернемся к рассмотрению вопросов, связанных с использованием *базы данных SQLite* в приложениях под *Android*. Любая *база данных*, созданная в приложении доступна любому классу приложения, но недоступна из вне. Чтобы открыть *доступ* к базе данных другим приложениям необходимо использовать *контент-провайдеры* (*Content* Providers).

Для создания и обновления *базы данных* в *Android* предусмотрен *класс* SQLiteOpenHelper. При разработке приложения, работающего с базами данных, необходимо создать *класс*-наследник от SQLiteOpenHelper, в котором обязательно реализовать методы:

onCreate() - вызывается при первом создании базы данных;
onUpgrade() - вызывается, когда необходимо обновить базу данных.

По желанию можно реализовать метод:

onOpen() - вызывается при открытии базы данных.

В этом же классе имеет смысл объявить строковые *константы*, в ко-

торых определить названия таблиц и столбцов. Полученный *класс* позаботится об открытии *базы данных*, если она существует, или о создании ее в противном случае, а так же об обновлении*базы данных* в случае необходимости.

В *Android* предусмотрен *класс* для работы с базой данных SQLite напрямую, этот *класс* называется SQLiteDatabase и содержит методы:

openDatabase()	- позволяет открыть базу данных;
update()	- позволяет обновить строки таблицы базы данных;
insert()	- позволяет добавлять строки в таблицу базы данных;
delete()	- позволяет удалять строки из таблицы базы данных;
query()	- позволяет составлять запросы к базе данных;
execSQL()	- позволяет выполнять запросы к базе данных.

Для добавления новых строк в таблицу используется *класс* ContentValues, каждый *объект* этого класса представляет собой одну строку таблицы и выглядит как ассоциативный *массив* с именами столбцов и значениями, которые им соответствуют.

Для получения результатов запросов к базе данных используется *класс* Cursor, объекты этого класса ссылаются на результирующий набор данных, позволяют управлять текущей позицией в возвращаемом при запросе наборе данных.

Для предоставления доступа к данным для других приложений мож-

но использовать *контент-провайдеры* (ContentProvider). Любая *информация*, управляемая контент-провайдером адресуется посредством *URI*:

content://authority/path/id

где:

content://	- стандартный требуемый префикс;
authority	- имя провайдера, рекомендуется использовать полное квалификационное имя пакета для избежания конфликта имен;
path	- виртуальная папка внутри провайдера, которая определяет вид запрашиваемых данных;
id	- первичный ключ отдельной запрошенной записи, для запроса всех записей определенного типа этот параметр не указывается.

Контент-провайдеры поддерживают стандартный *синтаксис* запросов для чтения, изменения, вставки и удаления данных.

Подробнее работу с SQLite базами данных в приложениях под *Android* рассмотрим в первой части лабораторной работы в этой теме.

17.2 Анимация

Android предоставляет мощные *API* для анимации элементов пользовательского интерфейса и построения *2D* и *3D* изображений.

Платформа *Android* предоставляет две системы анимации: *анимация* свойств, появившаяся в *Android* 3.0, и *анимация* компонентов пользовательского интерфейса (наследников класса View). Рассмотрим подробнее обе эти системы.

Анимация свойств (Property Animation). Система анимации свойств позволяет определить анимацию для изменения любого свойства объекта, независимо от того изображается оно на экране или нет. Используя эту систему, можно задать следующие характеристики анимации:

- **Продолжительность** предполагает задание длительности временного промежутка выполнения анимации, по умолчанию это значение равно 300 мс.
- **Временная интерполяция** предполагает вычисление значения свойства в каждый момент времени, как функции от промежутка времени, прошедшего с начала анимации.
- **Количество повторов и поведение** определяет необходимость повторения анимации при достижении конца заданного временного промежутка, а также количество повторов в случае необходимости. Эта же характеристика позволяет задать возможность воспроизведения в обратном порядке, если эта возможность выбрана, то анимация прокручивается вперед-назад заданное число раз.
- **Группа анимаций** позволяет организовать анимации в некоторое

множество и задать режим исполнения: одновременно, последовательно непрерывно или с некоторыми задержками.

- **Частота обновления кадров** определяет, как часто будет происходить смена кадров анимации. По умолчанию обновление происходит каждые 10 мс, однако скорость, с которой приложение сможет обновлять кадры, в конечном итоге, зависит от загруженности системы.

Большая часть *API* системы анимации свойств находится в пакете **android.animation**. Также можно использовать блоки интерполяции, определенные в пакете **android.view.animation**.

Класс Animator предоставляет базовую структуру для создания анимации. Напрямую этот *класс* обычно не используется, так как обеспечивает минимальную функциональность, поэтому чаще всего используются классы-наследники, расширяющие возможности класса Animator. Рассмотрим основные классы, используемые для создания анимации свойств.

- ValueAnimator (потомок класса Animator). Этот класс является главным обработчиком распределения времени для *анимации свойств*, а также рассчитывает значения свойства, предназначенного для анимации. Он обеспечивает всю основную функциональность: рассчитывает значения анимации и содержит распределенные во времени детали каждой анимации; содержит информацию о необходи-

мости повторений анимации; содержит слушателей, получающих уведомления о событиях обновления; предоставляет возможность задавать пользовательские типы для вычисления. В процессе *анимации свойств* можно выделить две части: вычисление значения свойства, для которого определяется анимация, и присвоение полученного значения соответствующему полю объекта.

ValueAnimator не выполняет вторую часть, поэтому необходимо следить за обновлениями значений, вычисляемых в классе ValueAnimator, и изменять объекты, подверженные анимации. Наглядно рассмотренные части анимации с использованием класса ValueAnimator представлены на рис. 2

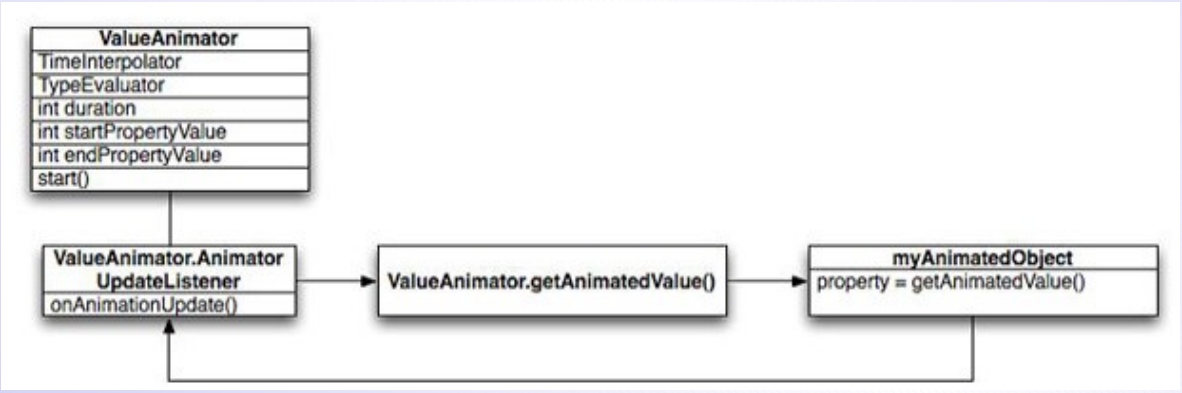


Рисунок 2. Процесс анимации свойств с использованием класса ValueAnimator

- AnimatorSet (потомок класса Animator). Предоставляет механизмы группировки анимаций, таким образом, что они выполняются некоторым образом относительно друг друга. Можно определять выполнение анимаций одновременно, последовательно и с временными задержками.

Классы-вычислители определяют как вычислять значения заданных свойств. Они получают: данные о *распределение времени*, предоставляемые классом Animator, начальное и конечное значения свойства, после чего на основе этих данных вычисляют значения свойства, для которого выполняется *анимация*. В системе *анимации свойств* существуют следующие вычислители:

- IntEvaluator для вычисления целочисленных значений свойств;
- FloatEvaluator для вычисления вещественных значений свойств;
- ArgbEvaluator для вычисления значений цвета в шестнадцатеричном представлении;
- TypeEvaluator - интерфейс, позволяющий создавать собственных вычислителей.

Интерполяторы определяют с помощью каких функций от времени, вычисляются значения свойств, для которых задается *анимация*. Интерполяторы определены в пакете *android.view.animation*. Если ни один

из существующих интерполяторов не подходит, можно создать собственный, реализовав *интерфейс* **TimeInterpolator**.

Подробнее с системой анимации свойств можно познакомиться по ссылке: здесь.

Анимация компонентов пользовательского интерфейса. Эта система может быть использована для реализации анимации преобразований над наследниками класса View. Для расчета анимации преобразований используется следующая *информация*: начальная точка, конечная точка, размер, поворот и другие общие аспекты анимации. *Анимация* преобразований может выполнять серии простых изменений содержимого экземпляра класса View. Например, для текстового поля можно перемещать, вращать, растягивать, сживать текст, если определено фоновое изображение, оно должно изменяться вместе с текстом. Пакет **android.view.animation** предоставляет все классы, необходимые для реализации анимации преобразований.

Для задания последовательности инструкций анимации преобразований можно использовать или *XML*, или *Android* код. Более предпочтительным является *определение* анимации в *XML* файлах, располагаться эти файлы должны в папке **res/anim/** проекта. *XML файл* должен иметь единственный *корневой элемент*, это может быть любой из отдельных элементов: <alpha>, <scale>, <translate>, <rotate>, *интерполятор*, или же элемент <set>, который содержит группы этих элементов, в том числе может содержать другие элементы <set>. *По*

умолчанию инструкции анимации выполняются одновременно, чтобы задать последовательное *исполнение* необходимо определить *атрибут* startOffset.

Подробнее с системой анимации преобразований можно познакомиться по ссылке: здесь.

Дополнительно к рассмотренным системам анимации может использоваться, кадровая *анимация*, которая реализуется быстрой сменой кадров, каждый *кадр* является графическим ресурсом и располагается в папке **res/drawable/** проекта.

Подробнее с кадровой анимацией можно познакомиться по ссылке: здесь.

17.3 2D и 3D графика

При разработке приложения важно четко понимать требования к графике в этом приложении. Для разных графических задач необходимы разные техники их решения. Далее в лекции рассмотрим несколько способов изображения графических объектов в *Android*.

Холсты и графические объекты. Платформа *Android* предоставляет *API* для изображения 2D графики, который позволяет изображать на холсте свои графические объекты или изменять существующие. Для отображения 2D графики существуют два пути:

1. Изобразить графику или анимацию в элементе пользовательского

интерфейса. В этом случае графика управляется процессом отображения иерархии элементов интерфейса. Подходит, когда необходимо отобразить простую графику, не требующую динамических изменений.

2. Изображать графику напрямую на холсте (класс Canvas). В этом случае необходимо позаботиться о вызове метода onDraw(), передавая его в класс Canvas, или же о вызове одного из draw...() методов класса Canvas (например, drawPicture()). Действуя таким образом, можно управлять анимацией. Этот путь подходит, когда необходимо постоянно перерисовывать окно приложения, например, для видео игр.

Аппаратное ускорение. Начиная с *Android* 3.0 (*API* уровень 11), конвейер изображения 2D графики в *Android* поддерживает аппаратное ускорение. Это означает, что все *операции* рисования на холсте исполняются с использованием *GPU*. В связи с увеличением требований к ресурсам *приложение* будет потреблять больше *RAM*. Аппаратное ускорение доступно *по* умолчанию, если целевой уровень *API* больше или равен 14, но может быть включено явно. Если в приложении используются только стандартные представления и *графика*, включение аппаратного ускорения не должно привести к каким-либо нежелательным графическим эффектам. Однако из-за того, что аппаратное ускорение поддерживается не всеми операциями 2D графики, его включение может нарушать неко-

торые пользовательские изображения или вызовы рисования. Проблемы обычно проявляются в невидимости некоторых элементов, появлении исключений или неверно изображенных пикселях. Чтобы исправить это, *Android* позволяет включать или выключать аппаратное ускорение на разных уровнях: уровень приложения, уровень активности, уровень окна, уровень элемента интерфейса.

OpenGL. *Android* поддерживает высокопроизводительную 2D и *3D* графику с использованием открытой графической библиотеки OpenGL, точнее OpenGL ES *API*. Библиотека OpenGL является кросс-платформенным *API*, который определяет стандартный программный *интерфейс* для аппаратного обеспечения, занимающегося обработкой *3D* графики. OpenGL ES является разновидностью OpenGL, предназначенной для встроенных устройств. *Android* поддерживает несколько версий OpenGL ES *API*:

- OpenGL ES 1.0 и 1.1 поддерживается Android 1.0 и выше;
- OpenGL ES 2.0 поддерживается Android 2.2 (API уровень 8) и выше;
- OpenGL ES 3.0 поддерживается Android 4.3 (API уровень 18) и выше.

Поддержка OpenGL ES 3.0 на реальном устройстве требует реализации графического конвейера, предоставленной производителем. Поэтому устройство с *Android* 4.3 и выше может не поддерживать OpenGL ES 3.0.

Подробнее с графикой в *Android* можно познакомиться по ссылкам: 1; 2; 3.

17.4 Основные принципы разработки игровых приложений для смартфонов

Разработка игр дело обычно благодарное т. к. в игры люди играли, играют и будут играть. Даже если результат работы не принесет особой прибыли, в любом случае он способен доставить радость детям и друзьям, да и о себе забывать не стоит. При этом сам процесс разработки способен серьезно повысить уровень мастерства особенно начинающего разработчика.

Если возникло острое желание создать именно игровое *приложение*, необходимо иметь в виду некоторые особенности: практически любая *игра* предполагает наличие сюжета, игры обычно отличаются эффектным графическим оформлением и обеспечивают определенный игровой процесс (геймплей). И эти моменты стоит хорошо продумать прежде, чем начинать *программирование*.

Сюжет игры состоит из последовательности событий. Необходимость сюжета больше всего зависит от жанра игры: в некоторых жанрах можно обойтись совсем без сюжета. Не стоит, как недооценивать, так и переоценивать важность сюжета, т. к. он является лишь одной из составляющих успеха игры. Решение о том нужен или ненужен сюжет в игре,

если нужен, то в какой мере и каким образом он будет выстраиваться, необходимо принимать взвешенно и до начала разработки.

При выборе способа графического оформления игры стоит иметь в виду, что использование *3D* графики серьезно усложнит процесс разработки, даже несложная *3D игра* отнимет очень много времени. В большинстве игр для мобильных устройств достаточно 2D графики, особенно в случае начинающего разработчика или команды таковых.

Следует учитывать ограниченные возможности мобильных устройств: сравнительно невысокая вычислительная *мощность*; ограниченный объем оперативной и дисковой памяти; небольшой размер и невысокое разрешение экрана; возможные проблемы, связанные с организацией передачи данных; ограниченный заряд аккумуляторных батарей.

Вопросы и задания для самоконтроля

Выполните пожалуйста этот тест

Литература

1. Мобильная платформа Android: пять лет истории [Электронный ресурс]/Мобильная платформа Android. - Режим доступа: http://itc.ua/articles/mobilnaya-platforma-android-pyat-let-istorii/. - Дата доступа: 9.08.2015.
2. Что это за Android? [Электронный ресурс]/ Что это за Android? - Режим доступа: http://android.com.ua/android_os.html. - Дата доступа: 9.08.2015.
3. Версии Android [Электронный ресурс]/ Версии Android. - Режим доступа: http://android-phones.ru/android/. - Дата доступа: 8.09.2015.
4. Подробный обзор Android 4.3 [Электронный ресурс]/ Подробный обзор Android 4.3. - Режим доступа: http://habrahabr.ru/post/188344/. - Дата доступа: 9.08.2015.
5. Архитектура операционной системы Android [Электронный ресурс] / Архитектура операционной системы Android. - Режим доступа: http://android-shark.ru/arhitektura-operatsionnoy-sistemyi-android/. - Дата доступа: 9.08.2015.
6. Майер, Р. Android 2 : программирование приложений для планшетных компьютеров и смартфонов/ Р.Майер. - Москва: Эксмо, 2011. - 672 с.

7. Android для новичков: что такое виджет? [Электронный ресурс] / Android для новичков: что такое виджет? - Режим доступа: http://www.androidpit.ru/chto-takoe-vidzhet. - Дата доступа: 9.08.2015.

8. Application Fundamentals [Электронный ресурс] / Application Fundamentals. - Режим доступа: http://developer.android.com/guide/components/fundamentals.html. - Дата доступа: 9.08.2015.

9. Введение в разработку для платформы Android [Электронный ресурс] / Введение в разработку для платформы Android. - Режим доступа: http://www.ibm.com/developerworks/ru/library/os-android-devel/. - Дата доступа: 9.08.2015.

10. Тидвелл, Дж. Разработка пользовательских интерфейсов / Дж. Тидвелл. - Москва: СПб, Питер, 2008. - 416 с.

11. Кронин, Д. Алан Купер об интерфейсе. Основы проектирования взаимодействия / Д. Кронин, А. Купер, Р. Рейман. - СПб.: Символ'Плюс, 2009. – 688 с.

12. Универсальное разрешение Android: идеально на всех экранах [Электронный ресурс] / Универсальное разрешение Android: идеально на всех экранах. - Режим доступа: http://habrahabr.ru/post/177093/. - Дата доступа: 9.08.2015.

13. Android M developer Preview [Electronic resource] / Android M developer Preview. - Mode of access: http://developer.android.com/index.html. -

Date of access: 9.08.2015.

14. Dialogs [Electronic resource] / Developer Android. - Mode of access: http://developer.android.com/guide/topics/ui/dialogs.html. - Date of access: 9.08.2015.

15. Программирование под Android / З. Медникс [и др.]. - СПб.: Питер, 2012. - 496 с.

16. [Азбука] Жизнь на кончиках пальцев. Тактильное взаимодействие с сенсорным экраном смартфона [Электронный ресурс] / CMS magazine. - 2013. - Режим доступа: http://www.cmsmagazine.ru/library/items/moblile/tactile-interaction/. - Дата доступа: 9.08.2015.

17. Особенности проектирования тачевых интерфейсов [Электронный ресурс] / Особенности проектирования тачевых интерфейсов. - Режим доступа: http://habrahabr.ru/post/150905/. - Дата доступа: 9.08.2015.

18. Реализация сенсорного интерфейса в новых и существующих играх [Электронный ресурс] / Реализация сенсорного интерфейса в новых и существующих играх. - Режим доступа: http://software.intel.com/ru-ru/node/394259. - Дата доступа: 9.08.2015.

19. Датчики и сенсоры современных мобильных устройств [Электронный ресурс] / INFOCITY. - 2013. - Режим доступа: http://www.infocity.az/?p=8233. - Дата доступа: 9.08.2015.

20. Введение в Gestures [Электронный ресурс] / Введение в Gestures. - Режим доступа: http://habrahabr.ru/post/120016/. - Дата доступа: 9.08.2015.

21. Android Research Blog [Electronic resource] / Android Research Blog. - Mode of access: http://androidresearch.wordpress.com/tag/gesture-builder/. - Date of access: 9.08.2015.

22. android.gesture [Electronic resource] / android.gesture. - Mode of access: http://developer.android.com/reference/android/gesture/package-summary.html. - Date of access: 9.08.2015.

23. Location and Sensors APIs [Electronic resource] / Location and Sensors APIs. - Mode of access: http://developer.android.com/guide/topics/sensors/index.html. - Date of access: 9.08.2015.

24. Location Strategies [Electronic resource] / Location Strategies. - Mode of access: http://developer.android.com/guide/topics/location/strategies.html. - Date of access: 9.08.2015.

25. Библиотеки [Электронный ресурс] / Сайт Александра Климова. - Режим доступа: http://developer.alexanderklimov.ru/android/library/. - Дата доступа: 9.08.2015.

26. Android Support Library [Electronic resource] / Support Library. - Mode of access: http://developer.android.com/tools/support-library/index.html#overview. - Дата доступа: 9.08.2015.

27. Библиотека NineOldAndroids [Электронный ресурс] / NineOldAndroids. - Режим доступа: http://nineoldandroids.com/. - Дата доступа: 9.08.2015.

28. Библиотека ActionBarSherlock [Электронный ресурс] / Библиотека ActionBarSherlock. - Режим доступа: http://actionbarsherlock.com/. - Дата доступа: 9.08.2015.

29. Библиотека Яндекс-метрика [Электронный ресурс] / yandex.ru. - Режим доступа: http://api.yandex.ru/metrica-mobile-sdk/. - Дата доступа: 9.08.2015.

30. Facebook SDK for Android [Electronic resource] / Facebook SDK for Android. - Mode of access: https://developers.facebook.com/docs/android. - Date of access: 9.08.2015.

31. Universal Image Loader for Android [Electronic resource] / Universal Image Loader for Android. - Mode of access: https://github.com/nostra13/Android-Universal-Image-Loader. - Date of access: 9.08.2015.

32. jsoup: Java HTML Parser [Electronic resource] / jsoup: Java HTML Parser. - Mode of access: http://jsoup.org/.- Date of access: 9.08.2015.

33. Android Holo ColorPicker [Electronic resource] / Android Holo ColorPicker. - Mode of access: https://github.com/LarsWerkman/HoloColorPicker. - Date of access: 9.08.2015.

34. MapNavigator [Electronic resource] / MapNavigator. - Mode of access: https://github.com/tyczj/MapNavigator. - Date of access: 9.08.2015.

35. AChartEngine [Electronic resource] / AChartEngine. - Mode of access: http://code.google.com/p/achartengine/. - Date of access: 9.08.2015.

36. Обзор Android-библиотек [Электронный ресурс] / Обзор Android-библиотек. - Режим доступа: http://www.androidviews.net/. - Дата доступа: 9.08.2015.

37. Ad Vulna: A Vulnaggressive (Vulnerable & Aggressive) Adware Threatening Millions [Electronic resource] / fireeye.com. - Mode of access: http://www.fireeye.com/blog/technical/2013/10/ad-vulna-a-vulnaggressive-vulnerable-aggressive-adware-threatening-millions.html. - Date of access: 9.08.2015.

38. Бурнет, Э. Привет, Android! Разработка мобильных приложений / Э. Бурнет, - СПб: Питер, 2012. - 256 с.

39. Понимание SQL (Understanding SQL). [Электронный ресурс] / Понимание SQL. - Режим доступа: http://www.sql.ru/docs/sql/u_sql/. - Дата доступа: 9.08.2015.

40. Animation and Graphics Overview [Electronic resource] / Animation and Graphics Overview. - Mode of access: http://developer.android.com/guide/topics/graphics/overview.html. - Date of access: 9.08.2015.

41. Основные принципы разработки игр [Электронный ресурс] / Основные принципы разработки игр. - Режим доступа: http://habrahabr.ru/post/188372/. - Дата доступа: 9.08.2015.

42. Место сценариста в команде разработки игр [Электронный ресурс] / Место сценариста в команде разработки игр. - Режим доступа: http://habrahabr.ru/company/mailru/blog/197152/. - Дата доступа: 9.08.2015.

43. Создание Игр Для Начинающих [Электронный ресурс] / 3dg.me. - Режим доступа: http://3dg.me/ru/gamedev/basics/sozdanie-igr-dlya-nachinayushchih. - Дата доступа: 9.08.2015.

44. Суровый геймдев на примере трех игр для Android [Электронный ресурс] / Суровый геймдев на примере трех игр для Android. - Режим доступа: http://habrahabr.ru/post/195828/. - Дата доступа: 9.08.2015.

45. 4pda.ru [Электронный ресурс] / 4pda.ru. - Режим доступа: http://4pda.ru/forum/lofiversion/index.php?t315915-0.html. - Дата доступа: 9.08.2015.

46. Блог Андрея Монахова [Электронный ресурс] / Блог Андрея Монахова. - Режим доступа: http://andmonahov.blogspot.se/2012/03/glsurfaceview.html. - Дата доступа: 9.08.2015.

47. Peter Wayner [Electronic resource] / InfoWorld. - Mode of access: http://www.itworld.com/software/383227/10-reasons-browser-becoming-universal-os. - Date of access: 9.08.2015.

48. HTML [Electronic resource] / ru.wikipedia. - Mode of access: http://ru.wikipedia.org/wiki/HTML. - Date of access: 9.08.2015.

49. Основы интернет-технологий: учебное пособие / А.А. Липницкий [и др.]. - Сев. (Арктич.) федер. ун-т. Архангельск: ИПЦ САФУ, 2013. – 365 с.

50. Node.js [Electronic resource] / Node.js. - Mode of access: http://ru.wikipedia.org/wiki/Nodejs. - Date of access: 9.08.2015.

51. Видеоролики по работе с Intel XDK [Электронный ресурс] / Видеоролики по работе с Intel XDK.- Режим доступа: http://software.intel.com/ru-ru/html5/tools#tab-panel-begin. - Дата доступа: 9.08.2015.