

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

Брянский государственный технический университет

**Утверждаю**

**Ректор университета**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_О.Н.Федонин**

**«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2017 г.**

**ПРОГРАММИРОВАНИЕ МОБИЛЬНЫХ СИСТЕМ**

**Знакомство со средой разработки**

**Методические указания**

**к выполнению лабораторной работы №1**

**для студентов очной формы обучения по направлениям подготовки 02.03.03 – «Математическое обеспечение и администрирование информационных систем», 09.03.01 – «Информатика и вычислительная техника» и 09.03.04 – «Программная инженерия»**

**Брянск 2017**

УКД 004.43

Программирование мобильных систем. Знакомство со средой разработки. [Текст] + [Электронный ресурс]: Методические указания к выполнению лабораторной работы №1 для студентов очной формы обучения по направлениям подготовки 02.03.03 – «Математическое обеспечение и администрирование информационных систем», 09.03.01 – «Информатика и вычислительная техника» и 09.03.04 – «Программная инженерия». – Брянск: БГТУ, 2017. – 27с.

Разработал:

Д.Н.Панус

ст.преп.

Рекомендовано кафедрой «Информатика и программное обеспечение» БГТУ (протокол №2 от 16.09.2016)

Научный редактор Д.А.Коростелев

Редактор издательства Л.И.Афонина

Компьютерный набор Д.Н. Панус

Темплан 2017 г., п.273

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписано в печать 23.09.17. Формат 60х84 1/16 Бумага офсетная. Офсетная печать. Усл.печ.л. 1,8. Уч.-изд.л. 1,8 Тираж 1 экз. Заказ Бесплатно.

Издательство Брянского государственного технического университета

241035, Брянск, бульвар 50-летия Октября, 7, БГТУ. 58-82-49.

Лаборатория оперативной полиграфии БГТУ, ул. Институтская, 16.

# Цель работы

Целью работы является практическое знакомство с возможностями проектирования приложений для ОС Android. Приобретение навыка работы со средой разработки, организацией ресурсов приложения, описания манифеста приложения. Получение навыков сборки приложений для установки на устройства средствами IDE и утилит входящих в состав SDK.

Продолжительность работы – 4 часа.

# Порядок выполнения работы

1. Ознакомится c выбранной средой разработки: Eclipse + ADT, Android Studio или IntelliJ IDEA. Разобраться в назначении инструментов, применяемых в разработке.
2. Ознакомится со средой запуска приложений OS Android в рамках лабораторной работы.
3. Выяснить возможности предоставляемые тестовой средой (телефон или эмулятор ОС Android).
4. Ознакомиться с программным стеком ОС Android в рамках лабораторной работы.
5. Ознакомится с видами приложений в ОС Android, классифицировать разрабатываемое в рамках лабораторных работ приложение.
6. Разобраться какие компоненты, используемые в ОС Android необходимо применить в данной лабораторной работе.
7. Ознакомиться со структурой файла манифеста приложения и описать в нем требуемый от приложения функционал, а также требования предъявляемые к аппаратуре.
8. Изучить жизненный цикл приложения в ОС Android, приоритеты приложений и состояния процессов. Выяснить, что такое стек вызова Activity и реализовать в приложении механику реагирования на события жизненного цикла.
9. Ознакомиться с правилами хранения ресурсов приложения. Реализовать в дереве проекта требуемую иерархию файловой системы.
10. Описать в виде XML требуемые ресурсы приложения.
11. Ознакомиться с назначением класса Application при реализации приложения для ОС Android.
12. Ознакомиться с возможностями класса Activity, Реализовать для каждого экрана свое собственное Activity.
13. Произвести сборку готового приложения в виде \*.apk файла приложения средствами IDE.
14. Произвести сборку приложения посредством вызова консольных утилит входящий в SDK. Продемонстрировать полученный результат преподавателю.

# Описание компонентов среды разработки

Среда разработки приложений для Android включает все необходимое для создания, тестирования и отладки программ. В состав скачиваемого комплекта входит:

* **API-платформы Android.** Ядро среды разработки — библиотеки API, которые обеспечивают программисту доступ к стеку платформы Android. Эти же самые библиотеки используются компанией Google для написания встроенных в Android приложений.
* **Инструменты разработки.** Исходный код можно преобразовать в исполняемые приложения для Android. В состав среды разработки входят инструменты, которые позволяют компилировать и отлаживать приложения.
* **Менеджер виртуальных устройств и эмулятор.** Эмулятор Android ‒ это полностью интерактивный эмулятор устройств, включающий несколько альтернативных вариантов интерфейса. Эмулятор запускается внутри виртуального устройства Android, которое моделирует аппаратную конфигурацию определенной модели телефона.
* **Полный набор документации.** Среда разработки включает расширенную справочную информацию с примерами кода, в которой описывается, что входит в каждый программный пакет и класс и как их можно использовать. В дополнение к этому в справочной документации содержится стартовый курс для начинающих, а также подробное описание основ разработки программ для Android.
* **Онлайн-поддержка.** Группы Google по адресу http://developer.android.com/resources/community-groups.html — это форумы разработчиков, на которых часто появляются сообщения от команд инженеров и специалистов компании Google. Ресурс StackOverflow по адресу http://www.stackoverflow.com/questions/tagged/android также стал популярным — здесь публикуются вопросы, посвященные Android.

Для тех, кто использует популярную среду разработки Eclipse, Android выпустил специальный плагин, который упрощает процесс создания проекта и тесно связывает Eclipse с эмулятором и отладочными инструментами Android.

# Программный стек Android

Программный стек Android состоит из элементов, подробное описание которых приводится ниже. Упрощенно их можно представить как комбинацию ядра Linux и набора библиотек C/C++, которые доступны в фреймворке приложения. Последний обеспечивает управление функционированием рабочей среды и приложений.

* **Ядро Linux.** Работу системных служб (драйверы устройств, управление процессами и памятью, питанием, безопасность, сетевые службы) обеспечивает ядро Linux. Оно также отвечает за уровень абстракции между аппаратной начинкой и остальной частью программного стека.
* **Библиотеки.** Android включает разнообразные системные библиотеки C/C++ (например, SSL и libc), которые работают поверх ядра. Среди них можно выделить:
* *библиотеку для работы с мультимедиа*, которая обеспечивает проигрывание аудио- и видеофайлов;
* *менеджер интерфейса*, отвечающий за управление отображением;
* *графические библиотеки*, такие как SQL и OpenGL, для работы с 2D- и 3D-графикой;
* *библиотеку SQLite*, обеспечивающую работу встроенных баз данных;
* *SSL и WebKit* для работы встроенного веб-браузера и обеспечения интернет-безопасности.
* **Рабочая среда Android.** Особенным телефон на платформе Android делает не столько мобильная версия ОС Linux, сколько рабочая среда Android. Она включает в себя:
* *Библиотеки ядра.* Хотя приложения для Android разрабатываются на языке Java, Dalvik — это не виртуальная Java-машина. Библиотеки ядра Android обеспечивают основную функциональность библиотек ядра Java, а также присущий Android уникальный функционал.
* *Виртуальная машина Dalvik.* Dalvik ‒ это виртуальная машина на основе регистров, которая оптимизирована таким образом, чтобы на устройстве можно было запускать несколько приложений одновременно. В ее основе ядро Linux, которое обеспечивает работу потоков и низкоуровневое управление памятью.
* *Фреймворк приложений.* Фреймворк включает набор классов, которые используются для разработки приложений. Он также предоставляет обобщенные абстрактные классы для доступа к оборудованию и обеспечивает управление пользовательским интерфейсом и ресурсами приложения.
* *Уровень приложений.* Все программы, как встроенные, так и сторонние, разрабатываются на уровне приложений с использованием одних и тех же библиотек API. Уровень приложений функционирует внутри рабочей среды Android, используя классы и службы, открытые для доступа на этом уровне.

# Архитектура Android приложения

Приложения в Android состоят из слабосвязанных компонентов, которые собираются воедино с помощью программного манифеста. Манифест ‒ файл, описывающий все компоненты приложения и способы их взаимодействия, а также метаданные, в том числе требования к платформе и аппаратной конфигурации. Базовые службы приложения, составляющие архитектуру всех приложений для Android, а также являющиеся основой фреймворка.

* **Менеджер активностей.** Контроль за жизненным циклом активностей (Activity), включая управление стеком активностей. Каждый экран приложения будет наследником класса Activity. Активности используют представления для формирования графического пользовательского интерфейса, отображающего информацию и взаимодействующего с пользователем. С точки зрения разработки под настольные платформы активность ‒ эквивалент Формы (Form).
* **Сервисы.** Невидимые двигатели приложения. Сервисные компоненты работают в фоновом режиме, запуская уведомления, обновляя источники данных и видимые активности. Используются для регулярных операций, которые должны продолжаться даже тогда, когда активности приложения не на переднем плане;
* **Представления.** Используются при создании пользовательских интерфейсов для активностей.
* **Менеджер уведомлений.** Обеспечивает работу унифицированных ненавязчивых уведомлений для пользователей.
* **Источники данных.** Позволяют приложениям открывать доступ к данным. Данные компоненты нужны для управления базами данных в пределах одного приложения и предоставления к ним доступа извне. Источники данных задействуются при обмене информацией между разными программами. Это значит, что можно настраивать собственные объекты ContentProvider, открывая к ним доступ из других приложений, а также использовать чужие источники, чтобы работать с данными, которые открыли для приложения внешние программы. Устройства под управлением Android содержат несколько стандартных источников, которые предоставляют доступ к полезным базам данных, включая хранилища мультимедийных файлов и контактной информации.
* **Менеджер ресурсов.** Обеспечивает отображение некодированных ресурсов, таких как текстовые строки или изображения.
* **Намерения.** Система передачи сообщений между приложениями. Используя намерения, можно транслировать сообщения на системном уровне или для конкретных активностей или сервисов. Тем самым диктуется необходимость выполнения заданных действий. После этого Android сам определит компоненты, которые должны обработать поступивший запрос.
* **Широковещательные приемники.** Компоненты, принимающие транслируемые намерения. Широковещательные приемники автоматически запускают программу, чтобы она могла ответить на принятое намерение. Благодаря этому данный механизм идеально подходит для создания приложений, использующих событийную модель.
* **Виджеты.** Визуальные программные компоненты, которые можно добавлять на домашний экран. Этот особый вид широковещательных приемников позволяет создавать динамические, интерактивные компоненты, которые пользователи могут встраивать в свои домашние экраны.
* **Уведомления.** Система пользовательских уведомлений. Позволяет сигнализировать о чем-либо, не обращая на себя внимание или не прерывая работу текущей активности. Механизм уведомлений лучше всего подходит для сервисов и широковещательных приемников, когда необходимо привлечь внимание пользователя. Например, принимая текстовое сообщение или входящий звонок, устройство оповещает пользователя, мигая светодиодами, воспроизводя звуки, отображая значки или показывая сообщения.

# Перед началом работы

Чтобы начать разработку собственных приложений для Android, понадобится установочный пакет Android SDK и средства разработки Java-приложений. Существуют версии SDK, Java и Eclipse для различных платформ ‒ Windows, MacOS и Linux ‒ таким образом, можно создавать приложения для Android на любой ОС. Инструменты разработчика и эмулятор функционируют на всех трех платформах.

Код приложений для Android пишется с соблюдением синтаксиса Java, библиотеки ядра Android поддерживают большинство функций ядра API Java. Перед исполнением проекты должны быть переведены в байт-код ВМ Dalvik.

Ограничения по питанию и памяти предъявляют особые требования к дизайну приложений. Кроме того, отдельные вещи, которые считаются само собой разумеющимися на компьютере или в Интернете, не будут аналогично работать на мобильном телефоне. Помимо аппаратных ограничений необходимо учитывать пользовательское окружение. Мобильные телефоны используются на ходу, поэтому приложения для мобильного устройства должны быть быстрыми, отзывчивыми и простыми для пользователя.

# Создание приложения для Android

### Создание нового проекта

Для создания нового проекта воспользуйтесь менеджером новых проектов Android.

1. Выберите пункты File - New – Project;
2. В списке типов приложений выберите Android Project и нажмите кнопку Next;
3. В появившемся окне необходимо указать информацию о новом проекте. В поле Project name укажите имя файла проекта. Поле Package name используется для указания названия пакета Java, параметр Create Activity ‒ для указания имени класса, который станет стартовой активностью. Поле Application name можно использовать для назначения имени программы. Min SDK Version позволяет задавать минимальную версию Android, с которой будет совместимо приложение;
4. После указания необходимой информации о проекте нажмите Finish. Если вами был выбран параметр Create Activity, плагин ADT создаст новый проект, включающий класс, происходящий от класса активности.

Таким образом, вместо пустого проекта в шаблоне по умолчанию будет реализован функционал базового приложения Hello World («Здравствуй, мир»). Перед редактированием кода проекта рекомендуется настроить запуск и отладку.

### Сохранение конфигурации запуска приложения

Конфигурация запуска включает настройки компиляции и отладки приложений. Они позволяют управлять следующими функциями:

* выбор проекта и Активности для запуска;
* управление виртуальным устройством и эмулятором;
* настройки ввода и вывода (включая параметры консоли по умолчанию).

Чтобы сохранить конфигурацию запуска для приложения Android:

1. В меню Eclipse выполните команду Run - Run Configurations или Debug Configurations.
2. В списке типов проектов выберите пункт Android Application, щелкните на нем правой кнопкой мыши и выберите New.
3. Укажите название для текущей конфигурации. Для каждого проекта может быть создано несколько конфигураций. Выбирайте понятные названия, чтобы идентифицировать тот или иной набор параметров.
4. Теперь можно приступать к настройке запуска приложения. Первая вкладка (Android) позволяет выбирать проект для компиляции, а также активность, которая будет запускаться во время компиляции (или отладки) приложения.
5. На вкладке Target можно выбрать виртуальное устройство по умолчанию, на котором будет запускаться приложение. Также можно использовать параметр Manual, при этом при каждой компиляции/отладке нужно выбирать устройство или AVD. На этой вкладке можно изменить сетевые настройки эмулятора, очистить пользовательские данные или отключить анимацию, которая проигрывается во время запуска виртуального устройства. В поле Command Line Options при необходимости можно указать дополнительные параметры запуска эмулятора.
6. На вкладке Common есть некоторые дополнительные настройки.
7. Нажмите кнопку Apply. Конфигурация для запуска приложения сохранится.

### Компиляция и отладка приложений

В меню Run выберите пункт Run или Debug ‒ для запуска приложения в соответствии с сохраненными настройками либо Run Configurations, либо Debug Configurations ‒ для выбора конкретной сохраненной конфигурации.

Если вы используете плагин ADT, компиляция или отладка вашего приложения будет происходить следующим образом:

* текущий проект скомпилируется в исполняемый файл платформы Android с расширением .dex;
* исполняемый файл и внешние ресурсы упакуются в файл пакета Android с расширением .apk;
* запустится выбранное виртуальное устройство (если вы выбрали AVD, и он в данный момент не запущен);
* приложение установится на целевое устройство;
* приложение запустится.

Для отладки используется отладчик Eclipse, в нем можно задавать точки остановки для отладки кода ваших программ.

Если все выполнено правильно, новая активность запустится в эмуляторе.

# Манифест Android-приложения

Любое приложение, создаваемое в Android, содержит файл манифеста, *AndroidManifest.xml*, который хранится в корневом каталоге проекта. Манифест позволяет описывать структуру и метаданные приложения, его компоненты и требования.

Манифест включает в себя узлы (теги) для каждого компонента (Активностей, Сервисов, Источников данных и Широковещательных приемников), из которых состоит приложение, и с помощью Фильтров намерений и полномочий определяет, каким образом они взаимодействуют друг с другом и со сторонними программами.

В манифесте предусмотрены атрибуты для указания метаданных (значков и визуальных стилей). Дополнительные узлы верхнего уровня можно использовать для описания настроек безопасности, модульных тестов (юнит-тестов), аппаратных и системных требований.

Манифест содержит корневой тег *<manifest>* с атрибутом *package*, который ссылается на пакет проекта. Как правило, этот тег также включает в себя атрибут *xmlns:android*, поддерживаемый системными узлами внутри файла.

Для задания текущей версии приложения используется атрибут *versionCode*, который задается в виде целого числа. Это внутреннее значение используется для сравнения версий программы. Атрибут *versionName* применяется для указания публичной версии, которая выводится для пользователей.

Типичный тег *<manifest>* показан во фрагменте кода:

<manifest xmlns:android=http://schemas.android.com/apk/res/android

package="com.my\_domain.my\_app"

android:versionCode="1"

android:versionName="0.9 Beta">

[ ... вложенные узлы манифеста ... ]

</manifest>

Тег *<manifest>* включает в себя узлы, описывающие программные компоненты, настройки безопасности, классы для тестирования и требования, из которых состоит ваше приложение. Ниже перечислены теги, доступные внутри узла *<manifest>:*

* **uses-sdk.** Позволяет задать минимальную, максимальную и целевую версии SDK, которые должны быть доступны на устройстве, чтобы приложение смогло правильно функционировать. Основываясь на версии SDK, которая поддерживается установленной платформой, и используя сочетание атрибутов *minSDKVersion, maxSDKVersion* и *targetSDKVersion*, можно ограничить круг устройств, способных запускать приложение.

Атрибут *minSDKVersion* указывает на минимальную версию SDK, содержащую API, которая используется в программе.

Атрибут *maxSDKVersion* позволяет определить самую позднюю версию, которую будет поддерживать разрабатываемое приложение. Приложение будет невидимым в Android Market для устройств, управляемых системой с более свежей версией. Устанавливать значение для этого атрибута рекомендуется только в том случае, если вы абсолютно уверены, что приложение не работает на платформе с версией, выше заданной.

Атрибут *targetSDKVersion* позволяет указать платформу, для которой разрабатывается приложение. Устанавливая значение для этого атрибута, вы сообщаете системе, что для поддержки этой конкретной версии не требуется никаких изменений, связанных с прямой или обратной совместимостью;

* **uses-configuration.** Теги *uses-configuration* используются, чтобы указать все механизмы ввода данных, поддерживаемые приложением. Можно задать любую комбинацию, содержащую следующие устройства:
* *reqFiveWayNav* ‒ укажите для этого атрибута значение true, если вам необходимо устройство ввода, поддерживающее навигацию вверх, вниз, влево, вправо, а также нажатие выделенного элемента; в эту категорию входят как трекболы, так и манипуляторы D-pad;
* *reqHardKeyboard* ‒ если приложению нужна аппаратная клавиатура, укажите значение true;
* *reqKeyboardType* ‒ позволяет задать тип клавиатуры ‒ nokeys, qwerty, twelvekey или undefined;
* *reqNavigation* ‒ если требуется устройство для навигации, укажите одно из следующих значений ‒ nonav, dpad, trackball, wheel или undefined;
* *reqTouchScreen* ‒ если приложению понадобится сенсорный экран, выберите одно из следующих значений — notouch, stylus, finger или undefined.

Можно задавать несколько поддерживаемых конфигураций, например, устройство с емкостным сенсорным экраном, трекболом и аппаратной клавиатурой (либо *qwerty*, либо *twelvekey*), как показано ниже:

<uses-configuration android:reqTouchScreen=["finger"] android:reqNavigation=["trackball"]

android:reqHardKeyboard=["true"] android:reqKeyboardType=["qwerty"/>

<uses-configuration android:reqTouchScreen=["finger"] android:reqNavigation=["trackball"]

android:reqHardKeyboard=["true"] android:reqKeyboardType=["twelvekey"]/>

* **uses-feature.** Одно из преимуществ Android ‒ широкий диапазон аппаратных платформ, на которых он может работать. Теги *uses-feature* используются, чтобы задать все необходимые приложению аппаратные возможности. Это предотвратит установку программы на устройства, которые не соответствуют аппаратным требованиям. На сегодняшний день аппаратные возможности предлагают следующие варианты:
* *android.hardware.*camera (если для работы приложения нужна аппаратная камера);
* *android.hardware.camera.autofocus* (если требуется камера с автоматической фокусировкой).

Также можно использовать тег *uses-feature*, чтобы задать минимальную версию OpenGL, которая требуется для работы приложения. С помощью атрибута *glEsVersion* указывается версия OpenGL ES в виде целого числа. Первые 16 бит соответствуют мажорной версии, а последние ‒ минорной.

* **supports-screens.** После первой волны устройств с экранами HVGA в 2009 году список аппаратов под управлением Android пополнился моделями с поддержкой WVGA и QVGA. С помощью тега *supports-screen* задаются экранные размеры, которые поддерживаются (и не поддерживаются) разрабатываемым приложением. Точные цифры будут варьироваться в зависимости от аппаратного обеспечения, но в целом соответствие размеров и разрешений экранов определяется следующим образом:
* *smallScreens* ‒ экраны с разрешением меньшим, чем обычное HVGA, как правило, речь идет о QVGA;
* *normalScreens* ‒ используется для описания экранов стандартных мобильных телефонов, как минимум HVGA, включая HVGA и WQVGA;
* *largeScreens* ‒ экраны больших размеров, значительно больше, чем у мобильного телефона;
* *anyDensity* ‒ значение true, если приложение способно масштабироваться для отображения на экране с любым разрешением.

В версии SDK 1.6 (API level 4) значения по умолчанию для каждого атрибута ‒ true.

* **аpplication.** В манифесте может присутствовать только один экземпляр данного тега. В нем используются атрибуты, содержащие метаданные для вашего приложения (включая его название, значок и визуальный стиль). Во время разработки атрибуту *debuggable* нужно установить значение true, чтобы активизировать режим отладки. Тег *<application>* также играет роль контейнера, который включает в себя узлы для активностей, сервисов, источников данных и широковещательных приемников, описывающих компоненты приложения.
* **аctivity.** Тег *<activity>* требуется для каждой Активности, которую отображает приложение. Атрибут *android:name* используется для указания имени класса активности. С помощью этих тегов добавляется главная активность, которая будет запускаться первой, а также остальные экраны и диалоговые окна, которые могут показываться. Попытка запустить активности без соответствующего описания в манифесте приведет к выбросу исключения. Каждый тег *<activity>* поддерживает вложенные узлы *<intent-filter>,* указывающие, какие именно намерения могут запустить активность.
* **service**. Как и в предыдущем случае, каждый класс сервиса должен иметь тег *service*. Теги *service* поддерживают вложенные узлы *<intent-filter>,* с помощью которых происходит латентное связывание.
* **provider.** С помощью этого тега указываются все источники данных в приложении. Источники данных используются для управления доступом к базам данных и для обмена информацией в рамках одной или нескольких программ.
* **receiver.** Добавляя в манифест тег *receiver*, можно зарегистрировать широковещательный приемник, не запуская при этом приложение. Широковещательные приемники отслеживают события на глобальном уровне: пройдя регистрацию, они начнут срабатывать при трансляции системой или приложением соответствующего намерения. Регистрируя их в манифесте, можно сделать этот процесс полностью анонимным. При трансляции соответствующего намерения приложение стартует автоматически, запуская зарегистрированный приемник.
* **uses-permission.** Теги *uses-permission* как часть системы безопасности описывают полномочия, которые, по мнению разработчика, нужны приложению для полноценной работы. Добавленные полномочия предоставляются пользователю до установки. Для использования многих стандартных сервисов в Android требуются полномочия (в частности, для действий, связанных с платными услугами и безопасностью, таких как телефонные звонки, прием SMS или использование геолокационных сервисов).
* **permission.** Сторонние приложения также могут указывать полномочия, прежде чем предоставлять доступ к общим программным компонентам. Чтобы ограничить доступ к компоненту приложения, нужно описать соответствующие полномочия в манифесте. Для этого необходимо использовать тег *permission*. Компоненты текущего приложения могут требовать полномочия с помощью атрибутов *android:permission*. Другие программы должны содержать в своем манифесте теги *uses-permission*, чтобы использовать эти защищенные компоненты. Внутри тега *permission* указывается уровень доступа, который обеспечивается данным полномочием (*normal*, *dangerous*, *signature*, *signatureOrSystem*). Также можно указать метку и внешний ресурс, содержащий описание и объяснение рисков, которыми сопровождается выдача этого полномочия.
* **instrumentation.** Классы, производные от *Instrumentation*, предоставляют фреймворк для тестирования программных компонентов во время их выполнения. Они содержат методы-перехватчики, с помощью которых отслеживаются работа программы и ее взаимодействия с системными ресурсами.

Подробное описание манифеста и всех вышеперечисленных тегов можно найти по адресу http://developer.android.com/guide/topics/manifest/manifest-intro.html.

Мастер создания проектов в составе ADT (New Project Wizard) автоматически добавляет файл с манифестом для каждого нового проекта.

# Жизненный цикл приложения в Android. Приоритеты приложений и состояния процессов

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

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

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

Порядок, в котором завершается работа процессов с целью освобождения ресурсов, определяется их приоритетами. Этот показатель берется из самого приоритетного компонента.

Если приоритет двух приложений одинаковый, первым будет закрыто то, которое дольше всего проработало с пониженным приоритетом. На приоритет процесса также влияют межпрограммные связи. Допустим, одно приложение зависит от Сервиса или Источника данных, которые предоставляются другим приложением. Из этого следует, что у второго приложения приоритет как минимум не ниже, чем у первого.

Важно, чтобы структура приложения была корректной и чтобы его приоритет соответствовал работе, которую оно выполняет. В противном случае приложение может закрыться во время выполнения каких-то важных действий.

Ниже приводится описание программных состояний:

* **Активные процессы.** Активные процессы (на переднем плане) содержат компоненты, взаимодействующие с пользователем. Android поддерживает их отзывчивость, освобождая дополнительные ресурсы. Как правило, таких процессов очень мало, и они закрываются в самую последнюю очередь. Активные процессы включают в себя:
* объекты Activity в активном состоянии, то есть те, которые находятся на переднем плане и отвечают на пользовательские события;
* Широковещательные приемники, обрабатывающие события с помощью методов onReceive;
* Сервисы, в которых запущены обработчики onStart, onCreate или onDestroy;
* Сервисы, предназначенные для работы на переднем плане.
* **Видимые процессы**. Видимые, но не выполняющиеся в данный момент процессы содержат Активности, которые отображаются на экране. Это происходит, когда Активность частично перекрыта (диалоговым окном или другой полупрозрачной Активностью). Процессов, которые выводятся на экран, очень мало, поэтому их работа прерывается только в крайнем случае, если не хватает ресурсов для активных приложений.
* **Процессы с запущенными Сервисами.** Это процессы, содержащие работающие Сервисы. Компоненты Service могут выполняться непрерывно и не должны иметь графического интерфейса. Поскольку фоновые Сервисы не взаимодействуют с пользователем напрямую, они получают немного меньший приоритет, чем видимые Активности. Наличие таких Сервисов выводит процесс на передний план и делает его преждевременное завершение возможным, только если потребуются ресурсы для активных или видимых приложений.
* **Фоновые процессы.** Процессы, не имеющие ни видимых Активностей, ни работающих Сервисов. Как правило, существует множество фоновых процессов, работа которых будет завершаться по принципу «последний запущенный закрывается первым», чтобы освободить ресурсы для приложений, работающих на переднем плане.
* **Холостые процессы.** Для улучшения общей производительности системы Android часто сохраняет в памяти приложения, которые завершили жизненный цикл. Android поддерживает этот кэш, чтобы уменьшить время повторного запуска программ. Работа таких процессов прерывается при необходимости.

# Класс Application в Android

Создание собственной реализации класса Application дает возможность:

* контролировать состояние приложения;
* передавать объекты между программными компонентами;
* поддерживать и контролировать ресурсы, которые используются в нескольких компонентах одного приложения.

Вместе с процессом программы создается экземпляр класса Application, который описан и зарегистрирован в манифесте. Исходя из этого класс Application по природе синглтон (singleton) и должен регистрироваться соответствующим образом, предоставляя доступ к методам и свойствам.

### Наследование и использование класса Application

В листинге показан каркас для наследования класса Application и реализации его в качестве синглтона.

import android.app.Application;

import android.content.res.Configuration;

public class MyApplication extends Application {

private static MyApplication singleton;

// Возвращает экземпляр данного класса

public static MyApplication getInstance() {

return singleton;

}

@Override

public final void onCreate() {

super.onCreate();

singleton = this;

}

Создав новый класс Application, нужно зарегистрировать его внутри тега *<application>* в манифесте.

При запуске приложения создается экземпляр реализации класса Application. Добавляйте новые свойства для хранения программного состояния и глобальных ресурсов, чтобы иметь возможность получить к ним доступ из компонентов приложения:

MyObject value = MyApplication.getInstance().getGlobalStateValue();

MyApplication.getInstance().setGlobalStateValue(myObjectValue);

Такой подход особенно эффективен при передаче объектов между слабосвязанными частями программы, а также для контроля за общими ресурсами и состоянием приложения.

### Переопределение событий, связанных с жизненным циклом приложения

Класс Application ‒ обработчики событий, отвечающие за создание и завершение работы приложения, поведение программы при нехватке памяти и изменениях конфигурации. Переопределяя эти методы, можно предусмотреть поведение приложения в подобных ситуациях.

Методы, которые обычно требуют переопределения:

* **onCreate.** Вызывается при создании приложения переопределяется для инициализации синглтона программы и создания и инициализации свойств, в которых хранятся состояния приложения или общие ресурсы.
* **onTerminate.** Может быть вызван при преждевременном завершении работы объекта Application. Если работа приложения прерывается ядром, чтобы освободить ресурсы для других программ, процесс завершится без предупреждения и без вызова метода onTerminate из объекта Application.
* **onLowMemory.** Предоставляет возможность хорошо оптимизированным приложениям освобождать дополнительную память, когда система работает в условиях ограниченных ресурсов. Как правило, вызывается в ситуациях, когда фоновые процессы уже закрыты, а памяти для приложений, находящихся на переднем плане, все еще не хватает. Этот метод переопределяют, чтобы очищать кэш или освобождать ненужные ресурсы.
* **onConfigurationChanged.** В отличие от объектов Activity приложение не перезапускается при изменениях конфигурации. Этот метод нужно переопределять, если разработчику необходимо обрабатывать изменения конфигурации на уровне приложения.

В процессе переопределения этих обработчиков всегда нужно вызывать одноименные методы родительского класса.

public class MyApplication extends Application {

private static MyApplication singleton;

public static MyApplication getInstance() {return singleton;}

@Override

public final void onCreate() {

super.onCreate();

singleton = this;

}

@Override

public final void onTerminate() {

super.onTerminate();

}

@Override

public final void onLowMemory() {

super.onLowMemory();

}

@Override

public final void onConfigurationChanged(Configuration newConfig) {

super.onConfigurationChanged(newConfig);

}}

# Детальный обзор Активностей в Android

При создании экранов с графическим интерфейсом наследуется класс Activity и используются Представления для взаимодействия с пользователем. Каждая Активность ‒ экран, который приложение может показывать пользователям. Чем сложнее приложение, тем больше экранов нужно.

Активности создаются для каждого экрана, который необходимо отобразить. Как минимум начальный экран, который отвечает за работу основного пользовательского интерфейса приложения, должен быть определен. Такой интерфейс часто дополняется второстепенными Активностями, предназначенными для ввода информации, ее вывода, предоставления дополнительных возможностей. Запуск новой Активности (или возврат из нее) инициирует перемещение между экранами. Большинство Активностей проектируются таким образом, чтобы занимать весь экран, есть возможность создавать полупрозрачные или плавающие диалоговые окна.

### Создание Активности

Чтобы создать новую Активность, необходимо унаследовать ее от класса Activity. Внутри этого класса необходимо определить пользовательский интерфейс и реализовать нужную функциональность. Базовый каркас для новой Активности показан ниже:

package com.paad.myapplication;

import android.app.Activity;

import android.os.Bundle;

public class MyActivity extends Activity {

/\*\* Вызывается при первом создании Активности. \*/

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

}}

Базовый класс Activity представляет собой пустой экран, который инкапсулирует всю работу по отображению окна. Пустая Активность не особо нужна, поэтому первое, что необходимо сделать, ‒ создать пользовательский интерфейс с помощью Представлений и разметки.

Представления ‒ элементы управления, которые отображают данные и обеспечивают взаимодействие с пользователем. Android предоставляет несколько классов разметки, наследников ViewGroup, которые могут содержать Представления для проектирования пользовательских интерфейсов. Чтобы назначить пользовательский интерфейс для Активности, внутри обработчика onCreate нужно вызвать метод setContentView, принадлежащий объекту Activity.

В этом фрагменте в качестве пользовательского интерфейса для Активности выступает элемент TextView:

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

TextView textView = new TextView(this);

setContentView(textView);

}

Разметку можно создавать как в коде программы, используя объекты ViewGroup, так и с помощью внешних ресурсов, передавая соответствующий идентификатор. Этот стандартный для Android подход показан в следующем фрагменте:

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

}

Чтобы использовать Активность в приложении, не забудьте зарегистрировать ее в манифесте.

### Жизненный цикл Активности

Четкое понимание жизненного цикла Активности важно для создания приложений, которые должны правильно управлять ресурсами и оставлять целостное впечатление.

Как уже говорилось ранее, приложения в Android не контролируют жизненный цикл своих процессов, система сама управляет всеми программами и, как следствие, Активностями внутри них.

Во время контролирования процесса (в том числе при завершении его работы) состояние принадлежащей ему Активности помогает системе определить приоритет родительского приложения. Приоритет приложения, в свою очередь, влияет на то, с какой вероятностью его работа (а также работа Активностей внутри него) будет прервана.

### Стеки (очереди) Активностей

Состояние каждой Активности определяется ее позицией в соответствующей очереди ‒ стеке (last-in-first-out) объектов Activity, запущенных в данный момент. При запуске новой Активности экран, находившийся на переднем плане, перемещается на вершину стека. Если пользователь вернется обратно, объект Activity закроется, следующая Активность в стеке перемещается вверх и становится текущей. Как уже говорилось, на приоритет приложения влияет его самая приоритетная Активность. Когда диспетчер памяти в Android решает, какую программу закрыть, чтобы освободить ресурсы, он использует этот стек для определения приоритетов приложений на основе их Активностей.

### Состояния Активностей

* **Активное.** Когда Активность на вершине стека, она видна и выходит на передний план, имея возможность принимать пользовательский ввод. Android будет пытаться любой ценой сохранить ее в рабочем состоянии, при необходимости прерывая работу других Активностей, которые находятся на более низких позициях в стеке, обеспечивая для нее все необходимые ресурсы. Ее работа будет приостановлена, если на передний план выйдет другая Активность.
* **Приостановленное.** В некоторых ситуациях ваша Активность будет видна на экране, но не сможет принимать пользовательский ввод: в этот момент она приостановлена. Такое состояние наступает, когда полупрозрачные или плавающие диалоговые окна становятся активными и частично ее перекрывают. В приостановленном виде Активность рассматривается как полноценно работающая, однако она не может взаимодействовать с пользователем. Ее работа может преждевременно завершиться, если системе нужно выделить ресурсы для той Активности, что на переднем плане. При полном исчезновении с экрана Активность останавливается.
* **Остановленное.** Когда Активность перестает быть видимой, она останавливается и остается в памяти, сохраняя всю информацию о своем состоянии. То есть становится кандидатом на преждевременное закрытие, если системе понадобится память для чего-нибудь другого. При остановке Активности важно сохранить данные и текущее состояние пользовательского интерфейса. Как только объект Activity завершает свою работу или закрывается, он становится неактивным.
* **Неактивное.** Это состояние наступает после того, как работа объекта Activity была завершена, и перед тем, как он будет запущен снова. Такая Активность удаляется из стека и должна запускаться повторно, чтобы ее можно было вывести на экран и снова использовать.

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

### Отслеживание изменений текущего состояния

Чтобы экземпляры Активностей могли реагировать на изменения текущего состояния, Android предоставляет набор обработчиков для событий, возникающих при переходе объекта Activity через стадии полноценный, видимый и активный.

В листинге приведен каркас с заглушками для методов внутри Активности, которые обрабатывают изменения состояний. Комментарии к каждой такой заглушке описывают действия, которые нужно учитывать при обработке этих событий.

package com.paad.myapplication;

import android.app.Activity;

import android.os.Bundle;

public class MyActivity extends Activity {

// Вызывается при входе в "полноценное" состояние.

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

// Инициализируйте Активность.

}

// Вызывается, когда метод onCreate завершил свою работу,

// и используется для восстановления состояния пользовательского

// интерфейса

@Override

public void onRestoreInstanceState(Bundle savedInstanceState) {

super.onRestoreInstanceState(savedInstanceState);

// Восстановите состояние UI из переменной savedInstanceState.

// Этот объект типа Bundle также был передан в метод onCreate.

}

// Вызывается перед тем, как Активность становится "видимой".

@Override

public void onRestart(){

super.onRestart();

// Загрузите изменения, учитывая то, что Активность

// уже стала "видимой" в рамках данного процесса.

}

// Вызывается в начале "видимого" состояния.

@Override

public void onStart(){

super.onStart();

// Примените к UI все необходимые изменения, так как

// Активность теперь видна на экране.

}

// Вызывается в начале "активного" состояния.

@Override

public void onResume(){

super.onResume();

// Возобновите все приостановленные обновления UI,

// потоки или процессы, которые были "заморожены",

// когда данный объект был неактивным.

}

// Вызывается для того, чтобы сохранить пользовательский интерфейс

// перед выходом из "активного" состояния.

@Override

public void onSaveInstanceState(Bundle savedInstanceState) {

// Сохраните состояние UI в переменную savedInstanceState.

// Она будет передана в метод onCreate при закрытии и

// повторном запуске процесса.

super.onSaveInstanceState(savedInstanceState);

}

// Вызывается перед выходом из "активного" состояния

@Override

public void onPause(){

// "Замораживает" пользовательский интерфейс, потоки

// или трудоемкие процессы, которые могут не обновляться,

// пока Активность не находится на переднем плане.

super.onPause();

}

// Вызывается перед тем, как Активность перестает быть "видимой".

@Override

public void onStop(){

// "Замораживает" пользовательский интерфейс, потоки

// или операции, которые могут подождать, пока Активность

// не отображается на экране. Сохраняйте все введенные

// данные и изменения в UI так, как будто после вызова

// этого метода процесс должен быть закрыт.

super.onStop();

}

// Вызывается перед выходом из "полноценного" состояния.

@Override

public void onDestroy(){

// Очистите все ресурсы. Это касается завершения работы

// потоков, закрытия соединений с базой данных и т. д.

super.onDestroy();

}

}

Как видно из кода, переопределяя обработчики, всегда нужно вызывать одноименные методы родительского класса.

### Классы Activity в Android

Android SDK включает набор классов, наследованных от Activity. Они предназначены для упрощения работы с виджетами, которые часто встречаются в обычном пользовательском интерфейсе. Ниже перечислены некоторые из них:

* **MapActivity.** Инкапсулирует обработку ресурсов, необходимых для поддержки элемента *MapView* внутри Активности;
* **ListActivity.** Обертка для класса *Activity*, главная особенность которой — виджет *ListView*, привязанный к источнику данных, и обработчики, срабатывающие при выборе элемента из списка;
* **ExpandableListActivity.** То же самое, что и *ListActivity*, но вместо *ListView* поддерживает *ExpandableListView*;
* **TabActivity.** Позволяет разместить несколько Активностей или Представлений в рамках одного экрана, используя вкладки для переключения между элементами.

**Задание на лабораторную работу**

Необходимо создать Android приложение, с одним экраном.

**Общие требования** при выполнении лабораторной работы:

1. Разобраться в процессах жизненного цикла приложений android.
2. Реализовать класс наследник *Application* с необходимыми для переопределения методами жизненного цикла.
3. Разобраться с этапами жизни экранов (*Activity* или *Fragment*) android.
4. Реализовать классы наследники *Activity* в приложении. Наследник *Activity* должен иметь необходимые обработчики для стандартных событий: *onCreate, onStart, onStop, onRestart, onPause, onDestroy* и т.д.
5. Главный экран должен иметь всю необходимую информацию для идентификации студента:

* 5 *TextView* элементов для отображения подписей к данным на экране: Фамилия, Имя, группа, email, номер телефона;
* 5 *TextView* элементов для отображения на экране данных о: Фамилия, Имя, группа, email, номер телефона;
* 1 *EditText*;
* 1 *Button*.

1. Для расположения элементов на экранах должны быть использованы менеджеры компоновки (не менее 2 различных).
2. Нажатие на *TextView* содержащий номер телефона, приводит к автоматическому переходу в приложение набора номера с отображением в нем номера телефона из *TextView*. (Реализовать по средствам XML разметки).
3. Нажатие на *TextView,* содержащий email, в Java коде должен быть задан полный шаблон письма содержащий поля Body, Mime, и т.д. В качестве адресата должен значится email из этого *TextView*. После формирования структуры email должен произойти старт приложения для его отправки при помощи механизма намерений (Intent) "android.intent.action.SENDTO".
4. Элемент *EditText* должен при получении фокуса открывать клавиатуру набора чисел.
5. Элемент *Button* должен содержать текст "Набрать студента". При клике на него должен происходить немедленный набор номера студента из *TextView,* содержащего номер телефона. (Делать через java код).
6. В рамках лабораторной работы: максимум требований должен реализовываться через XML разметку, за исключением случаев, когда требуется именно реализация через java код.
7. В файле манифеста должно быть запрошено у пользователя разрешение на прямой набор номера.
8. В приложении должны быть реализована собственная иконка, для всех вариантов dpi экранов, без использования генератора иконок IDE.
9. Приложение должно выводить через объект *Log* информацию о события жизненного цикла приложения activity и fragment, а также информацию обо всем интерактиве с пользователем;
10. Все текстовые константы и константные строки, в обязательном порядке должны быть вынесены в ресурсы приложения;
11. Разобраться с процессом сборки apk файла при помощи только утилит входящих в SDK, а также содержимым apk файла после компиляции. Gradle сборка ответом на вопросы преподавателя не является. Повторить процесс сборки приложения при помощи консоли.

**Контрольные вопросы**

1. Что такое программный стек Android?
2. Что такое виртуальная машина Dalvik и ART?
3. Как происходит сборка приложения средствами SDK? Из каких этапов состоит?
4. Каков состав APK файла? Зачем необходим каждый из компонентов в него входящий?
5. Зачем необходимо производить подпись приложения специализированным ключом?
6. Назовите основные используемые в работе SDK инструменты и для чего они необходимы?
7. Что такое манифест приложения Android. Зачем необходим?
8. Что описывается в файле манифеста? Его состав.
9. Класс Application. Наследование и применение.
10. События жизненного цикла приложения. 4 события.
11. Activity. Создание, описание, требования.
12. Жизненный цикл Activity.
13. Стек (Очередь) Activity.
14. Состояние Activity (4 состояния)
15. Отслеживание изменения состояния. Обработчики Activity.

# Список рекомендуемой литературы

1. Майер, Р. Android 2: Программирование приложений для планшетных компьютеров и смартфонов:[пер. с англ] / Р. Майер. - М.: Эксмо, 2011. - 672с.
2. Коматинени, С. Android 4 для профессионалов. Создание приложений для планшетных компьютеров и смартфонов:[пер. с англ]/ С. Коматинени, Д. Маклин. - М.:2012. 880с.
3. Медникс, З. Программирование под Android:[пер. с англ]/ З. Медникс, Л. Дорнин, Б. Мик, М. Накамура. - Спб.:Питер, 2012. - 496с.
4. Аndroid Developers. режим доступа: http://developer.android.com