

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

Брянский государственный технический университет

**Утверждаю**

**Ректор университета**

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

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

**Программирование МОБИЛЬНЫХ СИСТЕМ**

**Многоэкранные мобильные приложения, разработанные через оПисание нативным кодом java**

**Методические указания**

**к выполнению лабораторной работы №3**

**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 с базами данных и источниками данных в ОС Android. [Текст] + [Электронный ресурс]: Методические указания к выполнению лабораторной работы №5 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. – 16 с.

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**

Цель работы – приобретение навыка разработки интерфейсов мобильных приложений, при помощи программирования на нативном языке платформы - Java.

Продолжительность работы - 2 часа.

**Порядок выполнения работы**

1. Выяснить программные возможности, предоставляемые тестовой средой (телефон или эмулятор ОС Android).
2. Разобраться, какие компоненты, используемые в ОС Android необходимо применить в данной лабораторной работе.
3. Ознакомиться со структурой файла манифеста приложения и описать в нем требуемый от приложений функционал, а также требования, предъявляемые к аппаратуре.
4. Ознакомиться с правилами хранения ресурсов приложения. Реализовать в дереве проекта требуемую иерархию файловой системы.
5. Произвести сборку готового приложения в виде \*.apk файла приложения средствами IDE.
6. Продемонстрировать полученный результат преподавателю.

**Присваивания пользовательского интерфейса Активностям**

Новая Activity изначально – пустой экран, где вы должны разметить свой пользовательский интерфейс. Интерфейс можно описать не только с помощью XML-разметки, но и используя более традиционный способ – описание java-кодом. Ниже приведен пример создания пользовательского интерфейса:

|  |
| --- |
| package com.example.app;    import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.widget.TextView;    public class MainActivity extends AppCompatActivity {    @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  // создание TextView  TextView textView = new TextView(this);  // установка текста в TextView  textView.setText("Hello Android!");  // установка высоты текста  textView.setTextSize(22);  // установка визуального интерфейса для activity  setContentView(textView);  }  } |

Здесь весь интерфейс представлен элементом TextView, которое предназначено для выводa текста. С помощью методов, которые, как правило, начинаются на *set*, можно установить различные свойства *TextView*. Например, в данном случае метод *setText()* устанавливает текст в поле, а *setTextSize()* задает высоту шрифта.

Для установки элемента в качестве интерфейса приложения в коде Activity вызывается метод *setContentView()*, в который передается визуальный элемент.

Если используются Фрагменты для разделения интерфейса Активности, то Представление, которое загружается внутри обработчика *onCreate*, будет выглядеть как разметка, описывающая относительное местоположение всех Фрагментов (или их контейнеров). Как только Фрагмент загружается в Активность, все содержащиеся в нем Представления становятся частью иерархии этой Активности.

Можно использовать подобный подход, в то же время более оптимально определять визуальный интерейс в файлах xml, а всю связанную логику определять в классе *activity*. Тем самым можно достигнуть разграничения интерфейса и логики приложения, их легче будет разрабатывать и в последствии модифицировать.

**Описание разметки**

Описывая разметку через код программы, важно не забыть о параметрах *LayoutParameters*. Применить их можно с помощью метода *setLayoutParams* или во время вызова *addView*.

|  |
| --- |
| LinearLayout ll = new LinearLayout(this);  ll.setOrientation(LinearLayout.VERTICAL);  TextView myTextView = new TextView(this);  EditText myEditText = new EditText (this);  int lHeight = LinearLayout.LayoutParams.MATCH\_PARENT;  int lWidth = LinearLayout.LayoutParams.WRAP\_CONTENT;  ll.addView(myTextView, new LinearLayout.LayoutParams(lHeight, lWidth));  ll.addView(myEditText, new LinearLayout.LayoutParams(lHeight, lWidth));  setContentView(ll); |

**Получение элементов в коде и их идентификаторы**

Кроме текста, ширины и высоты текстовое поле устанавливает еще один важный атрибут – id. Знак + в записи android:id="@+id/header" означает, что если для элемента неопределен id со значением *header*, то его следует определить. Чтобы использовать этот файл в качестве основного интерфейса перейдем к *MainActivity* и изменим ее код:

|  |
| --- |
| package com.example.app;    import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.widget.TextView;    public class MainActivity extends AppCompatActivity {        protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);            // устанавливаем в качестве интерфейса файл second\_layout.xml          setContentView(R.layout.second\_layout);            // получаем элемент textView          TextView textView = (TextView) findViewById(R.id.header);          // переустанавливаем у него текст          textView.setText("Hello Android 7!");      }  } |

Для получения элементов по id класс *Activity* имеет метод *findViewById().* В этот метод передается идентификатор ресурса в виде R.id.[идентификатор\_элемента]. Этот метод возвращает объект *View* - объект базового класса для всех элементов, поэтому результат метода еще необходимо привести к типу *TextView*.

**Фрагменты**

Фрагменты позволяют разделять Активности на полностью независимые и пригодные для повторного использования компоненты, каждый из которых имеет свой жизненный цикл и пользовательский интерфейс.

Главное преимущество Фрагментов заключается в легкости, с которой можно создавать динамический и гибкий дизайн пользовательский интерфейсов, способных адаптироваться к экранам с разными размерами.

Для создания нового фрагмента необходимо наследовать класс *Fragment*, описать интерфейс и реализовать функциональность, которую он инкапсулирует (последние 2 действия необязательны).

Если фрагменту требуется пользовательский интерфейс, переопределите обработчик *onCreateView*, чтобы загрузить и вернуть нужную иерархию Представлений, как показано ниже:

|  |
| --- |
| package com.paad.fragments;  import android.app.Fragment;  import android.os.Bundle;  import android.view.LayoutInflater;  import android.view.View;  import android.view.ViewGroup;  public class myFragment extends Fragment{  @Override  Public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {  //Создайте или загрузите интерфейс фрагмента, а затем верните его.  //Если Фрагмент не содержит интерфейс, верните null.  Return inflater.inflate(R.layout.my\_fragment, container, false);  }  } |

В отличие от Активностей Фрагменты не нужно регистрировать в манифесте. Они могут существовать исключительно в рамках Активности, в которую были добавлены.

**Знакомство с Менеджером фрагментов**

Каждая Активность включает в себя Менеджер фрагментов для управления принадлежащими ей Фрагментами. Вы можете получить доступ к этому менеджеру с помощью метода *getFragmentManager*.

|  |
| --- |
| **FragmentManager fragmentManager = getFragmentManager();** |

Класс *FragmentManager* предоставляет методы для доступа к Фрагменту, который был добавлен в Активность, а также для действий, выполняемых с ними (добавление, удаление, замена).

**Стили**

Стиль — это один или несколько сгруппированных атрибутов форматирования, которые отвечают за внешний вид и поведение элементов или окна. Стиль может задавать такие свойства, как ширину, отступы, цвет текста, размер шрифта, цвет фона и так далее. Сами стили хранятся в XML-файлах, отдельно от файлов разметки.

Подобное разделение напоминает использование каскадных стилей CSS для веб-документов, которые также отвечают за стили HTML-элементов и хранятся в отдельных файлах.

Например,есть следующий код разметки для TextView:

|  |
| --- |
| <TextView  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:textColor="#00FF00"  android:typeface="monospace"  android:textSize="18sp"  android:text="@string/hello" /> |

Можно вынести все свойства в файл стилей следующим образом:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <resources>  <style name="MyTextStyle" parent="@style/Text">  <item name="android:textSize">18sp</item>  <item name="android:textColor">#00FF00</item>  <item name="android:typeface">monospace</item>  </style>  </resources> |

Тогда файл разметки теперь будет выглядеть так:

|  |
| --- |
| <TextView  style="@style/MyTextStyle"  ...  android:text="@string/hello" /> |

Мы удалили все свойства для текста из файла разметки и разместили их в файле стилей в ресурсе под именем MyTextStyle, который содержит теперь все необходимые свойства.

Чтобы создать файл со стилями, нужно создать новый XML-файл в папке *res/values/* вашего проекта. Имя файла не имеет значения, главное, чтобы расширение было XML, а сам файл находился в указанной папке. В проекте, создаваемом студией, уже есть готовый файл *res/values/styles.xml*, в который вы можете добавить новые стили. А также вы можете создать свой отдельный файл стилей.

Корневым узлом файла должен быть элемент *<resources>*. Для каждого элемента, которому требуется стиль, нужно добавить элемент *<style>* с уникальным именем. Далее создаются элементы *<item>* для каждого свойства и присваиваются им имена, которые отвечают за выбранное свойство. Значением элемента *<item>* должно выступать ключевое слово, цвет в шестнадцатиричном значении, ссылка на другой тип ресурсов или другое значение в зависимости от свойства стиля. Ниже представлен образец такого стиля:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <resources>  <style name="MyTextStyle" parent="@android:style/TextAppearance.Medium">  <item name="android:layout\_width">match\_parent</item>  <item name="android:layout\_height">wrap\_content</item>  <item name="android:textColor">#00FF00</item>  <item name="android:typeface">monospace</item>  </style>  </resources> |

Во время компиляции все свойства из файла стилей будут извлечены и применены к элементам.

Атрибут **parent** для элемента *style* является необязательным и позволяет задавать идентификатор ресурса другого стиля, из которого нужно наследовать свойства. При этом можно переопределить свойства в случае необходимости.

**Наследование стилей**

Наследование – мощный и полезный механизм, позволяющий использовать готовые проверенные наработки. С помощью атрибута **parent** в элементе **style** вы можете наследовать нужные свойства из существующих стилей, а также переопределить некоторые свойства или добавить свои дополнительные свойства. Предположим, мы решили наследоваться от существующего системного стиля Android для текстовых сообщений и слегка модифицировать его.

|  |
| --- |
| <style name="GreenText" parent="@android:style/TextAppearance">  <item name="android:textColor">#00FF00</item>  </style> |

Если вы собираетесь наследоваться от собственных стилей, используйте префикс имени наследуемого стиля перед создаваемым новым стилем, разделяя имена стилей точкой. Например, для создания нового стиля, который наследуется от стиля MyTextStyle, созданного нами ранее, где мы хотим получить красный текст, используйте следующий способ:

|  |
| --- |
| <style name="MyTextStyle.Red">  <item name="android:textColor">#FF0000</item>  </style> |

Теперь наш стиль наследует все свойства от стиля родителя, при этом мы изменили одно свойство *android:textColor*, чтобы текст выводился красным цветом. Вы можете ссылаться на новый стиль через конструкцию *@style/MyTextStyle.Red*.

**Намерения**

Намерение (Intent) — это механизм для описания одной операции - выбрать фотографию, отправить письмо, сделать звонок, запустить браузер и перейти по указанному адресу. В Android-приложениях многие операции работают через намерения.

Наиболее распространенный сценарий использования намерения - запуск другой активности в своём приложении. Но это не единственный вариант использования намерения. Также можно использовать для объявления о запуске активности или сервиса, направленных на выполнение каких-либо действий (как правило, речь о работе с определенной частью данных) или для передачи уведомлений о том, что произошло некое событие (или действие).

Намерения могут применяться для трансляции сообщений по системе. Любое приложение способно зарегистрировать широковещательный приёмник и отслеживать эти намерения с возможностью на них реагировать. Это позволяет создавать приложения, использующие событийную модель, в основе которой лежат внутренние, системные или сторонние события, передаваемые внешними программами.

**Неявные намерения**

Существует также неявный вызов активности. В этом случае при создании намерения мы используем не имя класса, а указываем параметры action, data, category с определёнными значениями. Система анализирует, что мы написали и начинает подбирать подходящий вариант для запуска.

Создавая новое неявное намерение для передачи в метод *startActivity()*, необходимо назначить действие, которое должно выполниться, а также при желании указать вспомогательный путь URI к тем данным, что нужно обработать. Вы также можете передать дополнительные данные в другую активность, используя параметр намерения *extras*.

При применении этого намерения для запуска активности система во время работы приложения сама найдёт класс компонента, который лучше всего подходит для заданного действия, учитывая указанный тип данных. Это значит, что вы можете создавать проекты, используя возможности других приложений и не зная при этом, как они называются и какую функциональность предоставляют.

Простой пример создания намерения с предопределенным действием **ACTION\_VIEW** для запуска браузера и перехода на нужный адрес:

|  |
| --- |
| Uri address = Uri.parse("http://developer.alexanderklimov.ru");  Intent openlinkIntent = new Intent(Intent.ACTION\_VIEW, address);  startActivity(openlinkIntent); |

В данном случае действие ACTION\_VIEW означает просмотр чего-либо. Мы указали нужные данные (адрес), и происходит запуск новой активности (браузера). При этом исходная активность приостанавливается и переходит в фоновый режим. Когда пользователь нажимает на кнопку Back, то он возвращается к исходной активности.

**Фильтры намерений**

Если намерение запрашивает выполнение какого-либо действия с определенным набором данных, то системе нужно уметь выбрать приложение (или компонент) для обслуживания этого запроса. На помощь приходят фильтры намерений, которые используются для регистрации активностей, сервисов и широковещательных приёмников в качестве компонентов, способных выполнять заданные действия с конкретным видом данных. С помощью этих фильтров также регистрируются широковещательные приёмники, настроенные на трансляцию намерением заданного действия или события.

Задействуя Фильтры намерений, приложения объявляют, что они могут отвечать на действия, запрашиваемые любой другой программой, установленной на устройстве. Чтобы зарегистрировать компонент приложения в качестве потенциального обработчика намерений, нужно добавить тег *<intent-filter>* в узел компонента в манифесте.

В фильтре намерений декларируется только три составляющих объекта Intent: действие, данные, категория. Дополнения и флаги не играют никакой роли в принятии решения, какой компонент получает намерение.

Например, в любом приложении есть главная активность, которая устанавливается как точка входа для задания:

|  |
| --- |
| <activity  android:name=".HelloWorldActivity"  android:label="@string/app\_name">  <intent-filter>  <action android:name="android.intent.action.MAIN"  <category android:name="android.intent.category.LAUNCHER"  </intent-filter>  </activity> |

Фильтр такого вида в элементе *<action>* помечает активность, как запускаемую по умолчанию. Элемент <category> заставляет значок и метку для деятельности отображаться на панели *Application Launcher*, давая пользователям возможность запускать задание и возвращаться к этому заданию в любое время после того, как оно было запущено.

Внутри узла Фильтра намерений можно использовать такие теги (и соответствующие атрибуты):

* **action** - использует атрибут *android:name*, чтобы задать название действия, которое будет обслуживаться. Каждый фильтр намерений должен иметь один (и только один) тег *action*. Действия должны быть представлены в виде уникальных строк, которые сами себя описывают. Рекомендуется выбирать названия, основываясь на соглашении об именовании пакетов в Java.
* **сategory** - использует атрибут android:name, чтобы указать, при каких обстоятельствах должно обслуживаться действие. Каждый тег *intent-filter* способен содержать несколько тегов *category*. Вы можете задать собственные категории или же брать стандартные значения, предоставляемые системой.
* **data** - этот тег дает возможность указать тип данных, с которым может взаимодействовать ваш компонент. При необходимости можно задать несколько тегов *data*.

При использовании метода *startActivity()* передаваемое неявное намерение, как правило, доходит лишь до одной активности. Если для выполнения заданного действия с указанными данными годятся сразу несколько активностей, пользователю предоставляется список выбора. Процесс, когда решается, какую активность лучше запустить, называется Утверждением намерений. Его цель — найти наиболее подходящий фильтр намерений. В целом весь алгоритм работает следующим образом.

Android собирает список всех доступных Фильтров намерений из установленных пакетов.

Фильтры, которые не соответствуют действию или категории Намерения, удаляются из списка. Совпадение происходит только в том случае, если Фильтр намерений содержит указанное действие (или если действие для него вовсе не задано). Совпадения не произойдет, только если ни одно из действий Фильтра намерений не будет эквивалентно тому, которое задано в Намерении. Для категорий процесс соответствия более строгий. Фильтр намерений должен включать в себя все категории, заданные в полученном Намерении. Фильтр, для которого категории не указаны, может соответствовать только таким же Намерениям (нет категорий).

Наконец, каждая часть пути URI из Намерения сравнивается с тегом *data* Фильтра намерений. Если в Фильтре указаны схема (протокол), сервер/принадлежность, путь или тип MIME, все эти значения проверяются на соответствие пути URI из Намерения. При любом несовпадении Фильтр будет удален из списка. Если в Фильтре намерений не указано ни одного параметра data, его действие будет распространяться на любые данные.

MIME — тип данных, который должен совпасть. При сравнении типов данных вы можете использовать маски, чтобы охватывать все подтипы (например, cats/\*). Если в Фильтре намерения указан тип данных, он должен совпасть с тем, который значится в намерении, при отсутствии тега data подойдет любой тип.

Схема — это протокольная часть пути URI, например http:, mailto: или tel:.

Имя сервера (или принадлежность данных) — часть URI между схемой и самим путем (например, www.google.com). Чтобы совпало имя сервера, схема Фильтра намерений также должна подойти.

После имени сервера идет путь к данным (например, /ig). Путь пройдет проверку только после схемы и имени сервера, содержащихся в теге.

Когда вы неявным образом запускаете Активность и вышеописанный процесс возвращает более одного совпадения, пользователю выводится список со всеми вариантами.

**Задание на лабораторную работу**

Необходимо доработать Android приложение, содержащее экраны для всех примененных, по заданию, особенностей пользовательского интерфейса, реализуя их при помощи кода на Java.

1. Все классы *activity* должны иметь в обязательном порядке обработчики для событий: *onCreate, onStart, onStop, onRestart, onPause, onDestroy.*
2. При наличии в оригинальном приложении системного или контекстного меня, оно обязательно реализуется. Элементы данного меню должны заполняться при помощи JAVA кода. При отсутствии в оригинальном приложении такого меню, остальные пункты требований реализуются в большем объеме с большим вниманием к деталям.
3. В рамках данной лабораторной работы реализуются при наличии в оригинальном приложении следующие элементы:
   1. Выпадающие списки выбора в *Application Bar*
   2. Интерактивные элементы в *Navigation Drawer*
   3. Любой пользовательский интерактив ориентированный на создание новый записей в списках
   4. Все списки с данными заполняются из кода случайными данными, подходящими по смыслу приложению (допустимо заполнять из предустановленных массивов данных)
   5. Работа с *Activity stack*, при его наличии, реализуется
   6. Все элементы интерфейса должны реагировать на действия пользователя как в оригинальном приложении.
   7. Любой пользовательский ввод который передается между экранами тоже должен передаваться между экранами.
4. Экраны, или их существенные части в рамках данной лабораторной работы создаются при помощи Java кода.  
   Под этим подразумевается:
   1. Создание View элементов наполняющих некоторый Layout и сам Layout через Java код.
   2. Данные элементы разметки должны полностью отсутствовать в XML разметке.
   3. Произвести стилизацию созданных элементов.
   4. Через код в итоге должно быть создано и стилизовано не менее 5 элементов
   5. Данное требование можно не оставлять в коде 4 и последующих лабораторных работ, но для сдачи данной работы обязательны к исполнению.
5. При отрисовке пользовательского интерфейса необходимо через Java код описать и применить к интерфейсу следующие Drawable объекты.
   1. Минимум один растровый Drawable под все плотности пикселей экрана (Иконка приложения к данному требованию не относится).
   2. XML Drawable объект.
6. При наличии в приложении какой-либо анимации, в приложении должны быть реализована через нативный Java код аналогичная анимация.
7. Приложении должно выводить через объект Log информацию о своей работе. Все обработчики пользовательских действий пишут о статусе вызова и исполнения
8. Все текстовые константы и константные строки, в обязательном порядке должны быть вынесены в ресурсы приложения.

**Контрольные вопросы**

1. Класс Application. Наследование и применение.
2. События жизненного цикла приложения. 4 события.
3. Activity. Создание, описание, требования.
4. Жизненный цикл Activity.
5. Стек (Очередь) Activity.
6. Состояние Activity (4 состояния)
7. Отслеживание изменения состояния. Обработчики Activity.
8. Из каких программных элементов строиться пользовательский интерфейс?
9. Что такое менеджер компоновки?
10. Основные менеджеры компоновки.
11. Основные view объекты, применяемые при построении интерфейсов.
12. События обработки взаимодействия с пользователем.
13. Toast уведомления. Их особенность и возможности по настройке отображения.

# Список рекомендуемой литературы

1. Майер, Р. Android 4. Программирование приложений для планшетных компьютеров и смартфонов. / Пер. с англ. – М.: Эксмо, 2013. – 816 С.
2. Коматинени, С. Android 4 для профессионалов. Создание приложений для планшетных компьютеров и смартфонов./ Коматинени С., Маклин Д. / Пер. с англ. – М.: Вильямс, 2012. – 880 С.